Rethinking Pruning for Accelerating Deep Inference At the Edge

Dawei Gao
SKLSDE & BDBC, Beihang University
david_gao@buaa.edu.cn

Xiaoxi He
ETH Zürich
hex@ethz.ch

Zimu Zhou
Singapore Management University
zimuzhou@smu.edu.sg

Yongxin Tong
SKLSDE & BDBC, Beihang University
yxtong@buaa.edu.cn

Ke Xu
SKLSDE & BDBC, Beihang University
kexu@nlsde.buaa.edu.cn

Lothar Thiele
ETH Zürich
thiele@ethz.ch

ABSTRACT

There is a growing trend to deploy deep neural networks at the edge for high-accuracy, real-time data mining and user interaction. Applications such as speech recognition and language understanding often apply a deep neural network to encode an input sequence and then use a decoder to generate the output sequence. A promising technique to accelerate these applications on resource-constrained devices is network pruning, which compresses the size of the deep neural network without severe drop in inference accuracy. However, we observe that although existing network pruning algorithms prove effective to speed up the prior deep neural network, they lead to dramatic slowdown of the subsequent decoding and may not always reduce the overall latency of the entire application. To rectify such drawbacks, we propose entropy-based pruning, a new regularizer that can be seamlessly integrated into existing network pruning algorithms. Our key theoretical insight is that reducing the information entropy of the deep neural network outputs decreases the upper bound of the subsequent decoding search space. We validate our solution with two state-of-the-art network pruning algorithms on two model architectures. Experimental results show that compared with existing network pruning algorithms, our entropy-based pruning method notably suppresses and even eliminates the increase of decoding time, and achieves shorter overall latency with only negligible extra accuracy loss in the applications.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous computing*; • **Computing methodologies** → *Neural networks*.

KEYWORDS

Deep Learning; Sequence Labelling; Network Pruning; Automatic Speech Recognition; Name Entity Recognition

ACM Reference Format:
Dawei Gao, Xiaoxi He, Zimu Zhou, Yongxin Tong, Ke Xu, and Lothar Thiele. 2020. Rethinking Pruning for Accelerating Deep Inference At the Edge. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403058

1 INTRODUCTION

Accurate and real-time inference of user context at the edge (*e.g.*, via speech or videos captured by embedded microphones or cameras) enables various applications such as personal assistants (*e.g.*, Apple Siri, Google Assistant) and smart home appliances (*e.g.*, Amazon Echo, Apple HomePod). To achieve high accuracy, researchers and companies are adopting deep neural networks (DNNs) in many complex inference tasks such as speech recognition, language understanding, and computer vision. For instance, DNNs have been integrated into the speech recognition products of Microsoft [28] and Baidu [10]. Despite their excellent accuracy, DNNs are computation-intensive, leading to large latency if executed in-edge or on-device, where computation capability is limited [32].

One effective technique to accelerate deep neural networks is network pruning [26, 32]. It radically reduces the complexity of a DNN without severely sacrificing its inference accuracy by removing unimportant units (*e.g.*, neurons) in the network, which potentially reduces execution latency of the DNN [2, 16].

Although network pruning proves effective to speed up an *individual* DNN, we ask a different yet practical question: **Does network pruning always reduce the overall latency of the *entire* deep inference application?** A deep inference application refers to an end-to-end solution for an inference task where a DNN is employed as an intermediate data processing module. **Of our particular interest are applications for sequence labelling tasks to be running on low-end resource-constrained edge devices which often do not have GPUs.** One popular solution for sequence labelling tasks is to first use a DNN to encode the input sequence into some intermediate representations and then adopt a decoder to search for the most likely output sequence [22, 25, 27]. Such a *DNN+decoder* model architecture is common for a wide spectrum of ubiquitous data mining applications such as speech recognition [23, 31] and language understanding [12, 29]. For instance, many state-of-the-art Automatic Speech Recognition (ASR) systems [10, 23, 28, 31] utilize Recurrent Neural Networks (RNN) for acoustic modeling of input speech. The acoustic scores generated by these deep models are then fed into a decoder containing a beam search process to decode the most likely sequence of words. Variants of *DNN+decoder* data processing pipeline are also typical in language understanding tasks *e.g.*, Name Entity Recognition (NER) [12, 29]. Common in the applications above, their end-to-end latency consists of both the delay to generate intermediate representations by the DNN (*DNN inference time*) and that to generate

KDD '20, August 23–27, 2020, Virtual Event, CA, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403058

the most likely output sequence via a maximum likelihood search decoding like beam search (*decoding time*).

Through a measurement study, we observe that existing network pruning algorithms [2, 16] may not always accelerate the *DNN+decoder* architecture, although such an architecture is widely used in many sequence labelling tasks such as ASR and NER. With the increasing degree of pruning, the DNN inference time decreases steadily, yet the decoding time tends to increase. Depending on the proportion between DNN inference time and the subsequent decoding time, the end-to-end latency may even exhibit a "U" shape with an increasing degree of pruning. The phenomenon is observed when applying different network pruning methods [2, 16] to models of various applications [12, 23, 29].

To mitigate the negative impacts of pruning on the overall latency of DNN-enabled sequence labelling applications, we propose entropy-based pruning, a new regularizer that can be plugged into existing pruning algorithms. Our key insight is that explicitly reducing the *information entropy* of the DNN outputs results in the decrease of the search space of beam search, and thus the decoding time. Particularly, we theoretically prove that the search space of the beam search decoder is upper bounded by a monotonically increasing function of the entropy of the DNN outputs.

We evaluate the performance of our method on different models in two representative applications: ASR [23] and NER [12, 23]. Experimental results show that our entropy-based pruning method is able to suppress and even avoid the increase of decoding time, thus achieving considerably shorter overall latency compared with network pruning without using our regularizer.

The main contributions of this work are summarized as follows.

- We show that existing pruning algorithms (*e.g.*, [2, 16]) may not reduce the end-to-end latency (DNN inference time + decoding time) of many DNN-enabled sequence labelling tasks (*e.g.*, ASR and NER) on low-end edge devices. In particular, the decoding time in these applications tends to increase when the prior DNN is pruned. The phenomenon is observed in various representative DNNs.
- To avoid slowdown of decoding due to network pruning, we propose entropy-based pruning, a new regularizer that can be seamlessly integrated into existing pruning algorithms. Furthermore, the effectiveness of our solution is theoretically guaranteed. To the best of our knowledge, this is the first network pruning algorithm for deep inference acceleration from the end-to-end latency perspective.
- We validate the effectiveness of our method on two common sequence labeling tasks. Evaluations show that compared with network pruning without our regularizer, the decoding time is reduced by up to 1.6 times in ASR and 10.6 times in NER. The corresponding overall latency is also significantly shorter. The speedup is achieved with negligible extra accuracy loss in the applications.

In the rest of this paper, we show the negative impact of network pruning on sequence label tasks in Sec. 2, introduce our entropy-based pruning method and prove its effectiveness in Sec. 3, evaluate its performance in Sec. 4, review related work in Sec. 5, and finally conclude in Sec. 6.

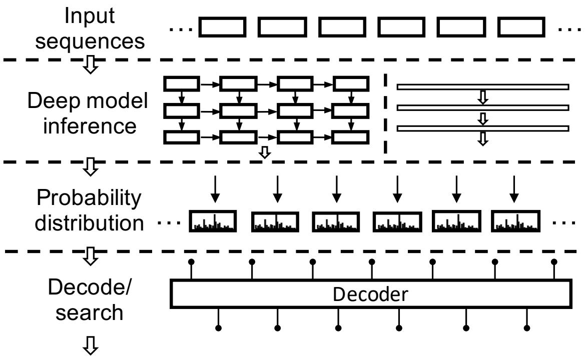


Figure 1: Typical data processing pipeline for sequence labeling tasks: A DNN (*e.g.*, RNN, CNN+RNN) is first used to encode the input sequence into hidden representations. Then a maximum likelihood search decoder is employed to generate the most likely output sequence.

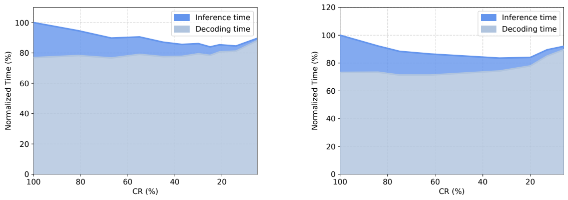
2 MOTIVATION STUDY

This section presents a measurement study on the side effect of pruning on the overall delay of two sequence labeling tasks.

2.1 Example Sequence Labeling Tasks of Deep Inference Applications

A widely used model architecture to solve sequence labeling tasks is *DNN+decoder*. Fig. 1 shows the data processing pipeline. The input sequence (*e.g.*, audio or video) is first encoded by a DNN into certain probability distribution. Then this probability distribution is fed into a decoder, which normally adopts beam search to generate the final output sequence. For ease of illustration, we base our measurements on two example sequence labeling tasks typical in many deep inference applications, *i.e.*, ASR and NER.

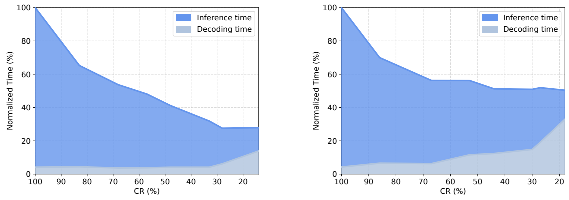
- **Automatic Speech Recognition (ASR)**. ASR is an essential building block for voice-based input in many intelligent assistant applications. It transforms speech audio into the corresponding textual data (*e.g.*, sentences in a natural language). In a typical ASR model architecture, deep models such as RNNs are used to correlate the input audio signals to phonemes. Afterwards, the decoder usually uses a beam search algorithm to find the most probable phonemes sequence, and then generates the most probable word sequence with the assistance of lexicon and language model [11]. The language model is usually predefined for a given language, which describes the transition probability of words based on prior grammatical and semantic knowledge. In this paper, we adopt the RNN model preset in pytorch-kaldi [23]. It takes the Mel-scale Frequency Cepstral Coefficients (MFCC) extracted from the speech as input, and consists of four Long Short-Term Memory (LSTM) layers. Each of the first three layers contains $1,024$ LSTM cells, and is activated by Rectified Linear units (ReLu). The output layer is activated by softmax. The decoder is implemented with WFST [18], which conducts beam search on a large graph consisted of the language model and lexicon.



(a) MNP

(b) VIBNet

Figure 2: Normalized overall latency of the preset pytorch-kaldi model for automatic speech recognition (LSTM + beam search) after increasingly pruning the network by (a) MNP and (b) VIBNet. Beam width is set to 13 as [21].



(a) MNP

(b) VIBNet

Figure 3: Normalized overall latency of a state-of-the-art model for name entity recognition (bi-LSTM-CRF + beam search) after increasingly pruning the bi-LSTM-CRF network by (a) MNP and (b) VIBNet. Beam width is set to 3.

- **Name Entity Recognition (NER)**. NER is a language understanding task crucial for interactions with voice assistant applications. It classifies the text (often recognized by ASR) into predefined categories (*e.g.*, part of speech) to facilitate further language mining or understanding. The input sentence of NER is typically transformed into vector representations. The vector representations are then fed into an RNN (a.k.a context encoder) to produce the encoded sequence. Conditional Random Fields (CRFs) model followed by search algorithms (usually beam search) is usually employed to assign labels to each word in the input sentence. In this work, we utilize the bi-LSTM-CRF model in [12, 29]. It contains a bidirectional LSTM layer with 1024 units activated by ReLu, a dense layer and a CRF layer. Then beam search is employed to generate the final label sequences.

2.2 Side Effect of Network Pruning on Overall Delay of Deep Inference Applications

We now show through measurements that although pruning speeds up the DNN, it slows down the subsequent beam search based decoding process. Consequently, the overall latency of the entire application may even increase after applying pruning to the DNNs.

Settings. We test two state-of-the-art DNN pruning strategies:

- **Magnitude-based Neuron Pruning (MNP) [16]**. It is one of the first structured pruning scheme for modern Convolutional Neural Networks (CNNs). MNP iteratively removes unimportant filters (importance usually assessed by L_1-norm) and retrains the network to recover its inference accuracy. In our work, we adapt the method to RNNs.
- **VIBNet [2]**. It is one of the best-performing structured network pruning methods. VIBNet removes unimportant neurons by reducing redundancy between adjacent layers and aggregating useful information into a sparse neuron subset.

We defer the model implementation details of these applications to Sec. 4. We measure the DNN inference time, the decoding time (via beam search), as well as the overall latency of each application on the NVIDIA Jetson TX2 platform [17]. To simulate the computation resources of low-end edge devices, the GPUs on the platform are disabled during measurements.

Observations. Fig. 2 and Fig. 3 show the normalized overall latency of ASR, NER, and LRR with the decrease of Compression Rate (CR). The numbers in each plot are normalized by the overall latency of the application without applying network pruning to the DNN. The compression ratio (CR) is calculated as the ratio between the number of parameters in the pruned network and that of the original network. We make the following observations.

- Although the DNN inference time in all the three applications is significantly reduced with the increase of pruning (a smaller CR), the overall latency does not necessarily decrease at the same pace, and can even increase when the DNN is heavily pruned. For example, when MNP prunes the DNNs used in ASR and NER to a compression ratio of 25% and 28%, the corresponding overall latency decreases to 83% and 27% of uncompressed baseline. However, when pruning the DNN from a compression ratio of 25% to 5% in ASR via MNP, the normalized overall latency contrarily increases from 83% to 90%. In NER, when compression ratio decreases from 28% to 14%, the overall latency still stays around 27%. Similar observations exist when adopting VIBNet to different DNNs.
- The reason behind the increase of the overall latency when the DNN is heavily pruned is that pruning tends to have an adverse impact on the decoding time of the beam search, the consumer of the DNN outputs. For example, when pruning the DNN in ASR using MNP / VIBNet, the decoding time remains stable when the compression ratio decreases from 100% to 40% / 50%, and increases to 114% / 122% if the DNN is pruned to a compression ratio of around 6%. In NER, the decoding time remains stable when the compression ratio is higher than 40%, but then increases to 332% when the compression ratio is around 14%. Note that ASR is slightly different from NER because it involves an extra effort to convert phones into words. Hence the ratio of decoding time vs. overall latency in significantly larger in ASR. Moreover, since the computation cost of this extra effort is not influenced by DNN, the percentage of increase in decoding time caused by network pruning is also lower compared with NER.

Summary. Network pruning may not accelerate the popular model architectures (DNN + decoder) used in sequence labeling tasks (*e.g.*, ASR and NER). The reason is that pruning on the DNN tends to increase the decoding time of the beam search afterwards. Depending on the portion of DNN inference time and the subsequent decoding time via beam search, the overall delay of the corresponding application may exhibit a "U" shape with an increasing degree of network pruning. The phenomenon is observed when applying different network pruning methods to different applications.

3 METHOD

In this section, we propose entropy-based pruning, an effective solution to eliminate the negative impact of network pruning on the overall latency of sequence labelling tasks.

3.1 Understanding DNN+Decoder Architecture

Before introducing our method and prove its effectiveness, we first explain the *DNN+decoder* model architecture in sequence labelling tasks in a more formal manner. Table 1 summarizes important notations that will be used throughout this section.

3.1.1 Sequence Labelling Task. We explain a sequence labelling task from a probabilistic perspective.

Given a common probability space $(\Omega, \mathcal{F}, P)$, an *input stochastic process* is a stochastic process $\{X_t\}$ with $1 \le t \le T$ defined on this probability space, in which each random vector X_t takes values in the M-dimensional real valued state space $\mathbb{R}^M$. A *labelling stochastic process* is another stochastic process $\{Y_t\}$ defined on $(\Omega, \mathcal{F}, P)$, in which each random variable Y_t takes value from a finite alphabet $\mathcal{Y}$ for the output labels. Denote N as the size of $\mathcal{Y}$, *i.e.*, $N = |\mathcal{Y}|$. We use the corresponding lowercase letter to denote a realisation of an stochastic process. For example, the realisation of an input stochastic process $\{X_t\}$ is a sequence of M-dimensional real numbers, denoted by $\mathbf{x} = \{x_i | 1 \le i \le T\} = \{X_i(\omega) | 1 \le i \le T\}$ with $\omega \in \Omega$.

A *sequence labelling task* is to assign a sequence of labels (output sequence), drawn from a fixed and finite alphabet $\mathcal{L}$, to a sequence of input data (input sequence) [7]. To solve the sequence labelling task, we train a sequence labelling algorithm $f : \mathbb{R}^{M \times T} \longrightarrow \mathcal{Y}^T$ which takes sequences $\mathbf{x} = \{x_1, x_2, \cdots, x_T\}$ as input and outputs labelling sequences $\mathbf{y} = \{y_1, y_2, \cdots, y_T\}$.

3.1.2 DNN+Decoder Architecture. DNN+decoder is widely used to solve sequence labelling tasks [22, 25, 27]. For ease of presentation, we denote a sub-sequence of the first t elements in $\mathbf{x}$ as $\mathbf{x}^{|t|} = \{x_1, x_2, \cdots, x_t\}$, and the sub-sequence of the first t elements in $\mathbf{y}$ as $\mathbf{y}^{|t|} = \{y_1, y_2, \cdots, y_t\}$. Note that $\mathbf{x}^{|T|} = \mathbf{x}$ and $\mathbf{y}^{|T|} = \mathbf{y}$.

The DNN+decoder architecture consists of two components:

- The *DNN* inputs vector sequences $\mathbf{x}$ and outputs vector sequences $\mathbf{p} = \{\mathbf{p}_t\}$. The vectors in sequences $\mathbf{p}$ are $\mathbf{p}_t = \{p_{t,i} | 1 \le i \le N\}$. Here $p_{t,i} = p(x_t | \mathbf{x}^{|t-1|}, \mathbf{y}^{|t-1|}, y_t = i)$ is the probability of observation x_t at time t, given all the previous observations $\mathbf{x}^{|t-1|}$ and the states $(\mathbf{y}^{|t-1|}, y_t = i)$. Here $y_t = i$ means "the state at time step t is the i-th state in the alphabet $\mathcal{Y}$".
- The *decoder* takes sequences $\mathbf{p} = \{\mathbf{p}_t\}$ from DNN as input, and aims to find a labelling sequence $\mathbf{y}$ with the largest $p(\mathbf{y}|\mathbf{x})$. This can be computed based on the Bayesian formula. Specifically, for a given input sequence $\mathbf{x}$, we have $p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})}$, where $p(\mathbf{x})$ is fixed for the input sequence $\mathbf{x}$. Thus the recursion formula becomes $p(\mathbf{x}^{|t|}, \mathbf{y}^{|t|}) = p(\mathbf{x}^{|t-1|}, \mathbf{y}^{|t-1|})$ $p(y_t | \mathbf{x}^{|t-1|}, \mathbf{y}^{|t-1|})$ $p(x_t | \mathbf{x}^{|t-1|}, \mathbf{y}^{|t|})$. Here $p(x_t | \mathbf{x}^{|t-1|}, \mathbf{y}^{|t|})$ is computed from the DNN; and $p(y_t | \mathbf{x}^{|t-1|}, \mathbf{y}^{|t-1|})$ is called the transition probability, which is often learned from prior knowledge like the language mode used for ASR [21] and CRF used for NER [12]. In the end, the decoder returns the sequence with the highest joint probability with $\mathbf{x}$, *i.e.*, $\max_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$. Normally a beam search is utilized to reduce the searching space.

3.1.3 Understand Overall Latency. The overall delay of a sequence labelling task adopting the DNN+decoder architecture consists of the inference time and the decoding time.

- The *inference time* is often highly correlated to the model size of the DNN. A DNN pruned to a smaller compression ratio usually results in a shorter inference time.
- The *decoding time* is related to the search space of beam search. At each time step t, the input of beam search is the reserved sequences $S_{t-1} = \{\mathbf{y}_1, \cdots, \mathbf{y}_{|S_{t-1}|}\}$ at time $t-1$ and the DNN output vector $\mathbf{p}_t$. Beam search generates $|S_{t-1}| \times N$ sequences by adding the possible states at time step t to the end of the sequences in S_{t-1}, and only those sequences with probabilities larger than a given threshold β are reserved in the sequence set S_t. Then S_t is passed to the $t+1$ step. Therefore, the decoding time at t is decided by the number of the reserved sequences at time $t-1$, *i.e.*, $|S_{t-1}|$.

Remarks. In the *DNN+decoder* architecture, the DNN and the decoder are connected via the probability distribution sequences $\mathbf{p}_t$. In particular, the DNN output $\mathbf{p}_t$ is fed into the decoder when deciding the reserved sequences at each time t and the decoding time is positively correlated to the size of the reserved sequences. We argue that the increase of decoding time due to network pruning observed in Sec. 2.2 can be avoided by reducing the *information entropy of* $\mathbf{p}_t$, as we will explain in the subsections below.

3.2 Entropy-based Pruning

Our key insight to avoid the increase of decoding time after network pruning is to explicitly regularizing the entropy of $\mathbf{p}_t$ during network pruning. Specifically, we propose to adopt the information entropy of the DNN output sequence $\mathbf{p}_t$ as a new regularizer:

$$L_e(\mathbf{p}) = \frac{1}{T} \sum_{\mathbf{p}_t \in \mathbf{p}} H(\mathbf{p}_t) = -\frac{1}{T} \sum_{\mathbf{p}_t \in p} \sum_{i}^{N} p_{t,i} \log p_{t,i} \qquad (1)$$

Consequently, the loss function of network pruning becomes

$$L(\mathbf{p}, \hat{\mathbf{p}}) = \tilde{L}(\mathbf{p}, \hat{\mathbf{p}}) + \gamma_e L_e(\mathbf{p}) \qquad (2)$$

where $\hat{\mathbf{p}}$ is the true labels, and $\tilde{L}(\mathbf{p}, \hat{\mathbf{p}})$ is the loss function specified by the application, *e.g.*, cross-entropy. The coefficient γ_e controls the strength of the regularizer. In principle, a larger γ_e encourages reduction of the decoding time, as we will show in Sec. 3.3 and Sec. 4. Yet an overly large γ_e may notably impair the accuracy of the application. We evaluate the impact of γ_e in Sec. 4.

Table 1: Summary of important notations.

Notation	Description
$\mathbf{x} = \{x_1, x_2, \cdots, x_T\}$	DNN input vector sequence
$\mathbf{y} = \{y_1, y_2, \cdots, y_T\}$	decoder output sequence
$\mathbf{x}^{\lvert t \rvert}$	sub-sequence of the first t elements in $\mathbf{x}$
$\mathbf{y}^{\lvert t \rvert}$	sub-sequence of the first t elements in $\mathbf{y}$
N	number of element in alphabet $\mathcal{Y}$
$\mathbf{p}_t = \{p_{t,i} \lvert 1 \le i \le N\}$	DNN output vector sequence; decoder input vector sequence
$S_t = \{\mathbf{y}_1, \cdots, \mathbf{y}_{\lvert S_t \rvert}\}$	sequences reserved in time step t
$\mathbf{s}_t(i, j)$	sequence generated by adding the j-th state to the end of the i-th sequence in S_{t-1}
β	threshold in beam search
$H(\cdot)$	information entropy
$B(\mathbf{p}_t, \beta)$	number of values in $\mathbf{p}_t$ that $\ge \beta$
k	adjustable coefficient to control network pruning

3.3 Theoretical Analysis on Effectiveness of Entropy based Pruning

We now theoretically prove that our entropy-based pruning, which explicitly enforces the decrease of $H(\mathbf{p}_t)$, is able to reduce the search space of beam search, and thus the decoding time.

3.3.1 Main Theoretical Claim. Recall from Sec. 3.1.3 that at time step t, the beam search needs to calculate the probability of each possible path sequence until this moment. There is in total $|S_{t-1}| \times N$ sequences to consider. Since N is the number of element in alphabet $\mathcal{Y}$ and is therefore fixed for a given sequence labelling task, the search space of the decoder is solely decided by $|S_t|$ with $1 \le t \le T-1$ for a given threshold β. Hence the effectiveness of our solution can be validated by the following claim: **reducing $H(\mathbf{p}_t)$ using our regularizer decreases the upper bound of $|S_t|$ given the threshold** β. Accordingly, the search space (and thus the decoding time) of the decoder is likely to be reduced.

3.3.2 Proofs of Theoretical Claim. The proof is divided in two parts.

- Reducing $H(\mathbf{p}_t)$ also reduces $H(\{\mathbf{s}_t(i, j)\})$, *i.e.*, the entropy of the distribution of sequences until time t. $\mathbf{s}_t(i, j)$ denotes the sequence generated by adding the j-th $(1 \le j \le N)$ state to the end of the i-th $(1 \le i \le |S_{t-1}|)$ sequence in S_{t-1}.
- Reducing $H(\{\mathbf{s}_t(i, j)\})$ decreases the tight upper bound of S_t given the threshold β.

The first part of the proof is guaranteed by the following lemma.

Lemma 1. The upper bound of $H(\{\mathbf{s}_t(i, j)\})$ is a strictly increasing function of $H(\mathbf{p}_t)$.

Proof. See Appendix A. □

For the second part of the proof, we first consider in general M-dimensional normalised vectors $\mathbf{v} \in \mathbb{R}^M$, $v_i \ge 0$ with $\sum_{i=1}^{M} v_i = 1$. $\mathbf{v}$ represents a discrete probability distribution with alphabet size M. The information entropy of the underlying distribution of $\mathbf{v}$ is denoted as $H(\mathbf{v}) = \sum_{i=1}^{M} v_i \log(1/v_i)$. The number of elements of $\mathbf{v}$ larger or equal than some constant $\beta \in \mathbb{R}$, $0 < \beta \le 1$ is defined as $B(\beta) = |\{i \mid 1 \le i \le M \wedge v_i \ge \beta\}|$. We are interested in $B(\beta)$ as a function of $H(\mathbf{v})$. As $\mathbf{v}$ is non-unique for a given $H(\mathbf{v})$ and $M \ge 2$, we determine tight upper bounds for $B(\beta)$, *i.e.*, $B(\beta) \le U_t(H(\mathbf{v}), \beta)$ for all $\mathbf{v}$ with entropy $H(\mathbf{v})$.

Lemma 2. The number of vector elements $B(\mathbf{v}, \beta)$ larger or equal than some constant β is bounded by $B(\mathbf{v}, \beta) \le U_t(H(\mathbf{v}), \beta)$, where $U_t(H(\mathbf{v}), \beta) = \min\{M, \lfloor 1/\beta \rfloor, \lfloor H(\mathbf{v})/(\beta \log(1/\beta)) \rfloor\}$ for $\beta \le 0.5$. If $\beta > 0.5$ we find $U_t(H(\mathbf{v}), \beta) = 1$ if $H(\mathbf{v}) \le (1-\beta)\log((M-1)/(1-\beta)) + \beta log(1/\beta)$ and $U_t(H(\mathbf{v}), \beta) = 0$ otherwise. The bound is tight.

Proof. See Appendix B. □

With the help of Lemma 2 we can now show the relationship between $H(\{\mathbf{s}_t(i, j)\})$ and $|S_t|$, as stated by the following lemma.

Lemma 3. The tight upper bound of $|S_t|$ given β is a monotonically increasing function of $H(\{\mathbf{s}_t(i, j)\})$.

Proof. See Appendix C. □

Remarks. Combining Lemma 1 and Lemma 3, we have that $|S_t|$ is upper bounded by a monotonically increasing function of $H(\mathbf{p}_t)$. Therefore, reducing $H(\mathbf{p}_t)$ will in general reduces $|S_t|$, and the search space of beam search, which accelerates the decoding.

4 EVALUATION

In this section, we evaluate the performance of our entropy-based pruning method on two data mining applications: ASR and NER. We first introduce the general experimental setup, and then present the specific settings and results for each individual application. Finally we summarize the findings of our evaluation.

4.1 General Experimental Setup

This subsection presents the common experimental settings, including the baseline algorithms, evaluation platform and metrics.

4.1.1 Compared Algorithms. We choose the same network pruning algorithms in Sec. 2 as baselines and apply our method to them. We compare the performance of the following four algorithms:

- **MNP**: It is the magnitude based neuron pruning algorithm for CNNs [16] adapted to support RNNs. It iteratively prunes the DNN. Denote the number of pruning iterations as n.
- **VIBNet**: It is one of best-performing neuron pruning algorithm [2] that removes neurons based on the variational information bottleneck. In VIBNet, a coefficient γ_{kl} is used to

control the strength of the information bottleneck. Normally a large γ_{kl} leads to a smaller CR.

- **MNP-Ent**: It applies our entropy-based pruning regularizer to the MNP pruning algorithm. We set γ_e as $\gamma_e = kn$, where k is an adjustable coefficient.
- **VIBNet-Ent**: It applies our entropy-based pruning regularizer to the VIBNet pruning algorithm. We set γ_e as $k \log \frac{\gamma_{kl}}{0.3}$ in ASR and set γ_e as a a constant k in NER.

4.1.2 Evaluation Platform. The experiments are conducted with Python 3.6 and Tensorflow 1.13. All the DNNs used in our experiments are trained (and pruned) in a workstation with 64GB memory, Intel Xeon Gold 5118 CPU and Quadro P5000 GPU. After training, all the applications (DNN+decoder) are tested on Jetson Tx2 [17] with 8G memory and ARM Cortex-A57 CPU. To simulate low-end edge devices, the GPUs on the platform are disabled.

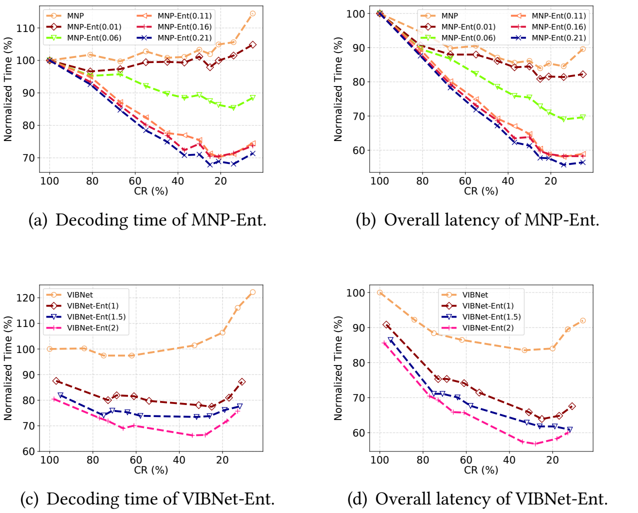
4.1.3 Evaluation Metrics. We quantify the performance of different algorithms using metrics from the the following aspects:

- **Latency**. We are interested in the impact of different pruning algorithms on the *overall latency* and the *decoding time* in each application. As in Sec. 2, the overall latency is normalized by the overall latency before applying pruning to the corresponding DNN. The decoding time is also normalized by the decoding time before network pruning.
- **Accuracy**. The adoption of network pruning should not induce severe accuracy loss in the target application. The accuracy metric is application-specific. For ASR, we adopt word error rate (WER) as the accuracy metric. It is calculated as $WER = 100 \cdot \frac{S+D+I}{N'}\%$, where S, D and I are the number of substitution, deletion and insertion, and N' is the total number of the words. For NER, we use error rate (ER) as the metric. It is calculated as $ER = 100 \cdot (1 - \frac{C}{N'})\%$, where C is the number of the words that are correctly labeled and N' is the total number of the words.
- **Compression Ratio**. The same as Sec. 2.2, the compression ratio (CR) is calculated as the ratio between the numbers of parameters in the pruned and the original DNN. The compression ratio quantifies the degree of compression. A smaller CR means a larger degree of network pruning.

4.2 Performance on ASR

Dataset. We evaluate the performance of different algorithms on ASR using Librispeech [19], a large-scale English corpus. During training, we choose 100 hours of speech data, which contains about 360, 000 sentences with lengths varying from 100 to 3, 000. These sentences are pre-processed with the 3-gram language model in kaldi [21] to generate MFCC features and the corresponding labels. The input MFCC features have 39 dimensions, where the first- and second-order frame-to-frame differences are 13 dimensions each [5]. The features are classified into 3, 432 categories. See [21] for more details about the pre-processing and feature extraction.

Model Implementation. We experiment with the LSTM model described in Sec. 2.1 as the DNN and beam search in Sec. 3.1 as the decoder. The LSTM model is optimized by momentum optimizer with cross entropy loss function. The learning rate and momentum are set to 0.01 and 0.9. The batch size is set as 32, and we train for two epochs. We directly use the beam search implemented in kaldi for decoding. The default beam width is set to 13 as suggested [21]. We vary k in $[0, 0.01, 0.06, 0.11, 0.16, 0.21]$ for MNP-Ent, and $[0, 1.0, 1.5, 2.0]$ for VIBNet-Ent. Pruning methods using different k values are labeled with MNP/VIBNet-Ent(k). In particular, $k = 0$ (making $\gamma_e = 0$) means network pruning without our regularizer.



(a) Decoding time of MNP-Ent.

(b) Overall latency of MNP-Ent.

(c) Decoding time of VIBNet-Ent.

(d) Overall latency of VIBNet-Ent.

Figure 4: Normalized decoding time and overall latency in ASR with an increasing degree of pruning. MNP-Ent and VIBNet-Ent with different values of k are used to prune an RNN (LSTM) network [23]. Note that $k = 0$ means the corresponding pruning method without our regularizer.

Results. Fig. 4 plots the normalized decoding time and the normalized overall latency with different compression ratio on ASR. Fig. 4(a) and Fig. 4(c) show the normalized decoding time of MNP-Ent and VIBNet-Ent. For MNP-Ent, when k is set to 0.06 or larger, the normalized decoding time stops increasing after network pruning. As k increases to 0.21, entropy-based pruning decreases the normalized decoding time further by 43%. Similar results are observed for VIBNet, as the uptrend of decoding time is suppressed when $k = 1.5$. Fig. 4(b) and Fig. 4(d) show that corresponding normalized overall latency when applying entropy-based pruning to the two pruning algorithms. As is shown, entropy-based pruning is able to allow a notable reduction in the overall latency after network pruning. For example, when the DNN is pruned to a compression ratio of 5% by MNP, the normalized overall latency decreases to 56% when applying our method ($k = 0.21$), whereas the normalized overall latency is 90% without our method (*i.e.*, when $k = 0$).

Table 2 shows the corresponding test accuracy (measured by word error rate) after applying different network pruning methods to the DNN. Comparing MNP and MNP-Ent with the same compression ratio, the increase of WER when applying MNP-Ent is 0.45% on average. No significant accuracy loss is induced when using VIBNet-Ent either. Note that the results for VIBNet and VIBNet-Ent are not at exactly the same compression ratio. This is because

Table 2: Word error rate (%) of ASR when applying MNP, MNP-Ent, VIBNet and VIBNet-Ent for network pruning.

Name	CR(%) / k	82	67	55	45	37	30	25	21	14	5
MNP	0	15.31	16.26	15.31	14.56	14.18	13.99	15.12	14.37	14.18	17.77
MNP-Ent	0.01	15.88	15.12	16.07	13.8	14.37	14.37	14.37	15.5	13.42	16.82
	0.06	14.93	15.31	15.88	15.69	15.69	14.37	13.99	14.37	15.12	17.58
	0.11	15.88	15.88	16.26	16.26	14.93	15.31	14.56	15.88	15.53	17.84
	0.16	16.07	14.93	16.00	16.07	15.01	15.41	16.26	14.93	15.31	17.77
	0.21	16.45	16.26	16.64	15.69	14.93	13.61	16.07	15.31	15.12	18.71
VIBNet	0	CR(%)	100	85	75	62	34	20	13	7	
		WER	12.44	12.48	13.42	13.42	13.99	15.69	17.77	17.2	
VIBNet-Ent	1	CR(%)	74	69	61	55	31	25	17	11	
		WER	13.99	13.99	13.8	13.61	14.56	15.88	17.2	18.53	
	1.5	CR(%)	76	72	64	58	33	27	19	12	
		WER	13.23	12.67	13.42	14.18	14.56	15.12	18.3	18.34	
	2	CR(%)	78	73	67	61	34	28	18	13	
		WER	13.99	13.42	13.23	14.74	14.56	17.2	17.96	18.56	

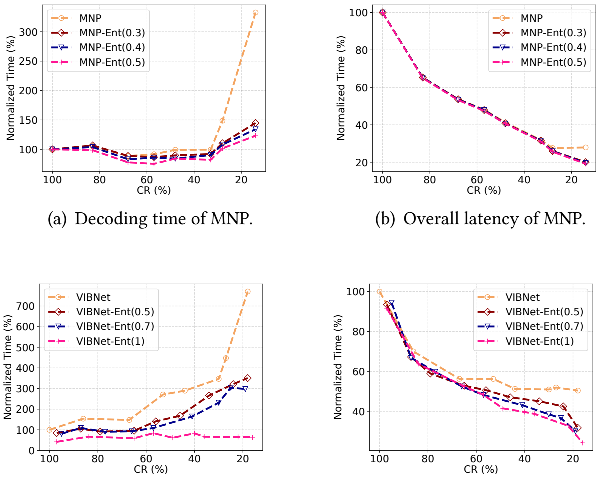
the compression ratio is indirectly controlled by γ_{kl} and thus difficult to be fine-tuned to a given value. In general, when applying our entropy-based pruning method to existing network pruning algorithms in ASR, only negligible extra accuracy loss is induced.

4.3 Performance on NER

Dataset. We use the English corpus in the CoNLL-2003 dataset [24], which was first used in a shared task of language-independent NER task in English and German. In the NER task in CoNLL-2003, the words (tokens) needs to be classified into 45 part-of-speech (POS) tags such as noun, verb, adverb etc. The training dataset of the English corpus contains 14, 987 sentences and 20, 3621 tokens, and the test dataset contains 3, 684 sentences and 46, 435 tokens.

Model Implementation. We experiment with the bi-LSTM-CRF model described in Sec. 2.1 as the DNN and beam search in Sec. 3.1 as the decoder. Before feeding into the DNN, each token is first embedded into a 300-dimensional vector with Glove [20], a popular word-to-vector library pre-trained using the Wikipedia 2014 and Gigaword 5 datasets. The DNN is ptimized by Adam [14] for 30 epochs in total, where the learning rate starts with 0.001 and decays by 0.9 each epoch. For decoding, a C++ implemented beam search is adopted with a default beam width of 3. For MNP, k varies among $\{0, 0.3, 0.4, 0.5\}$, and for VIBNet k varies among $\{0, 0.5, 0.7, 1\}$. Similar with the experiments of ASR, $k = 0$ ($\gamma_e = 0$) means network pruning methods without our entropy-based pruning regularizer.

Results. Fig. 5 show the normalized decoding time and overall latency when applying different network pruning methods to the DNN in NER. As shown in Fig. (a) and Fig. (c), compared with network pruning without using our regularizer, the increase of decoding time is suppressed by 209% (for MNP) and 697% (for VIBNet) with our regularizer. In particular, when the compression ratio is smaller than 30%, the decoding time of MNP and VIBNet surges to 3 to 6 times of the decoding time before network pruning. In contrast, the decoding time using our regularizer remains relatively stable (especially with a larger k). As shown in Fig. (b) and Fig. (d), when applying our method to the network pruning algorithms, the overall latency is also significantly reduced.



(a) Decoding time of MNP.

(b) Overall latency of MNP.

(c) Decoding time of VIBNet.

(d) Overall latency of VIBNet.

Figure 5: Normalized decoding time and overall latency in NER with an increasing degree of pruning. MNP-Ent and VIBNet-Ent with different values of k are used to prune an RNN (bi-LSTM-CRF) network [12, 29]. Note that $k = 0$ means the corresponding pruning method without our regularizer.

Table 3 shows the corresponding error rate when applying different network pruning algorithms to the DNN in NER. Similar to the results for ASR, our entropy-based pruning method only introduces negligible increase in error rate (0.01% on average) compared with pruning without our method.

4.4 Summary of Results

We summarize the main results of Sec. 4.2 and Sec. 4.3 as follows.

- Our entropy-based pruning method is able to suppress and sometimes avoid the increase of decoding time, especially when the DNN is heavily pruned. Compared with network pruning without our method, the decoding time is reduced

Table 3: Error rate(%) of NER when applying MNP, MNP-Ent, VIBNet and VIBNet-Ent for network pruning.

Name	CR(%) k	83	69	58	48	34	28	14		
MNP	0	15.26	15.18	15.26	14.7	14.68	14.94	14.83		
MNP-Ent	0.3	15.49	15.2	15.06	15.04	14.68	14.52	14.66		
	0.4	15.47	15.31	14.98	15.17	14.95	14.89	14.66		
	0.5	15.38	15.16	15.13	15.01	14.64	14.62	14.78		
VIBNet	0	CR(%)	100	86	67	53	45	30	28	18
		ER	13.42	14.61	14.61	14.59	14.89	15.37	16	16.53
VIBNet-Ent	0.5	CR(%)	87	79	65	57	47	35	25	18
		ER	13.65	14.09	14.52	14.55	14.87	15.55	16.08	18.14
	0.7	CR(%)	88	77	66	58	41	30	25	19
		ER	14.35	14.04	14.95	15.11	16.12	16.88	16.76	18.63
	1	CR(%)	85	66	57	50	40	36	22	16
		ER	13.72	14.9	15.45	16.08	16.21	16.01	17.7	18.1

by up to 1.6 times in ASR and 10.6 times in NER. The overall latency is also considerably lower with our method: up to 1.5 times shorter in ASR and 2.1 times shorter in NER.

- Compared with pruning without our method, our entropy-based pruning regularizer only introduces negligible extra accuracy loss on average: 0.45% in ASR and 0.01% in NER.

5 RELATED WORK

Our work follows the emerging trend to enable deep inference at the edge [32]. We focus on network pruning, an effective *model compression* technique for DNN acceleration. The technique is orthogonal to *runtime optimization*, another category of research on efficient DNN execution.

Model Compression for DNN Acceleration. Deep neural networks are typically over-parameterized and their model size can be radically reduced without decreasing model accuracy [3]. Pruning is a popular model compression technique that eliminates unimportant operations (*e.g.*, weights, neurons, kernels) in the model [2, 16]. It can be combined with other techniques like quantization to further reduce the precision of operations [9]. We refer readers to [26] for a comprehensive overview on model compression.

A pruning method usually takes a pre-trained DNN as input, iteratively removes unimportant operations and retrains the model to recover its accuracy, and finally outputs a compact model without notable accuracy drop. Central in a pruning scheme is the criteria to determine the importance of an operation, *i.e.*, the impact of removing one operation on the model accuracy. Various importance metrics have been proposed, such as magnitude-based [9, 16], sensitivity-based [4], information theory based [2], etc. The granularity of pruning is also crucial. Neuron pruning [2, 16]) is preferable than weight pruning [4, 9]) because the latter requires special hardware to achieve the expected DNN speedup [8].

We base our measurements and evaluations on two state-of-the-art structured pruning methods: MNP [16], one of the first magnitude-based neuron pruning schemes, and VIBNet [2], one of the best-performing structured pruning proposals based on information theory. We show that although both pruning methods reduce the inference time of DNNs, the overall latency (inference time + decoding time) of the entire application may even grow. This is because the importance criteria in existing pruning methods do not explicitly account for the delay of the subsequent data processing modules (*e.g.*, Viterbi beam search in our context) in the application. In contrast, we propose the first-of-its-kind pruning method that avoids slowdown of the entire deep inference application.

Runtime Optimization for Efficient DNN Execution. Given an uncompressed or compressed DNN, its execution efficiency can be improved via various software runtime optimization [6, 13, 15] or hardware accelerators [1, 8]. Runtime optimization is common for accelerating DNN execution on commodity mobile and edge devices such that no extra hardware is needed.

Runtime optimization aims to improve the resource utilization when executing DNNs on resource-constrained devices. For example, the execution of a DNN can be partitioned between a cloud/edge server and an edge device to reduce its delay [13]. Certain intensive computation can be offloaded to mobile GPUs on high-end edge devices [6]. There are also generic solutions that schedule DNN execution among heterogeneous computation resources and devices [15]. Our focus is orthogonal to runtime optimization. The impact of pruning on the overall delay of deep inference applications is agnostic to the resource utilization of the underlying DNN.

Our work is inspired by [30], which points out the side effect of pruning on the overall latency of DNN-based speech recognition applications. Compared with [30], our work differs in two aspects: *(i)* We conduct a more comprehensive study of the impact of pruning on the overall latency of deep inference applications. We experiment with various more complex DNNs (feed-forward networks, RNNs) and different applications (automatic speech recognition, name entity recognition). Conversely, the study in [30] only explores a simple feed-forward network and one application. *(ii)* We devise a new pruning method that avoids slowdown of these applications rather than a hardware accelerator of the subsequent beam search. Our solution is simple to implement, does not involve hardware modification, and has wider applicability.

6 CONCLUSION

In this work, we investigate the impact of network pruning for accelerating DNN-powered data mining applications on resource-constrained edge devices. We observe that while existing network pruning algorithms speed up an individual DNN, they tend to slow down the subsequent decoding process common in many sequence labelling tasks. Consequently, the end-to-end delay of the whole application may not necessarily decrease after DNN pruning. To mitigate this negative impact of network pruning, we design entropy-based pruning, a regularizer that can be plugged into existing network pruning algorithms. We theoretically prove the effectiveness of the new regularizer and evaluate its performance on two model architectures in different applications. Evaluations show that compared with network pruning without our regularizer, the decoding time is reduced by up to 1.6 times in ASR and 10.6 times in NER. The corresponding overall latency in these applications is also significantly shorter. We envision our work as a reference for future research on DNN compression from a holistic perspective.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable suggestions and comments. Dawei Gao, Yongxin Tong and Ke Xu's work was partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant No. 61822201, U1811463 and 71531001, and the Beijing Municipal Science and Technology Project under Grant Z191100002519012. Xiaoxi He and Lothar Thiele's work was supported in part by the Swiss National Science Foundation in the context of the NCCR Automation. Zimu Zhou's research was supported in part by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant. Zimu Zhou is the corresponding author.

REFERENCES

[1] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (2016), 105–112.

[2] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. In *Proceedings of International Conference on Machine Learning*. ACM, New York, NY, USA, 1143–1152.

[3] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. 2013. Predicting parameters in deep learning. In *Proceedings of Advances In Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2148–2156.

[4] Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Proceedings of Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 4860–4874.

[5] Sadaoki Furui. 1986. Speaker-independent isolated word recognition based on emphasized spectral dynamics. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 11. IEEE Press, Piscataway, NJ, USA, 1991–1994.

[6] Petko Georgiev, Nicholas D Lane, Cecilia Mascolo, and David Chu. 2017. Accelerating mobile audio sensing algorithms through on-chip gpu offloading. In *Proceedings of Annual International Conference on Mobile Systems, Applications, and Services*. ACM, New York, NY, USA, 306–318.

[7] Alex Graves. 2012. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*. Springer, 5–13.

[8] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of Annual International Symposium on Computer Architecture*. ACM, New York, NY, USA, 243–254.

[9] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of International Conference on Learning Representations*.

[10] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: scaling up end-to-end speech recognition. arXiv:1412.5567

[11] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[12] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv:1508.01991

[13] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: collaborative intelligence between the cloud and mobile edge. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, USA, 615–629.

[14] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of International Conference on Learning Representations*.

[15] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of International Conference on Information Processing in Sensor Networks*. ACM, New York, NY, USA, 1–12.

[16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. In *Proceedings of International Conference on Learning Representations*.

[17] Sparsh Mittal. 2019. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture - Embedded Systems Design* 97 (2019), 428–442.

[18] Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language* 16, 1 (2002), 69–88.

[19] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An ASR corpus based on public domain audio books. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*. IEEE Press, Piscataway, NJ, USA, 5206–5210.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*. 1532–1543.

[21] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi Speech Recognition Toolkit. In *Proceedings of Workshop on Automatic Speech Recognition and Understanding*. IEEE Press, Piscataway, NJ, USA.

[22] Rohit Prabhavalkar, Kanishka Rao, Tara N. Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. 2017. A Comparison of Sequence-to-Sequence Models for Speech Recognition. In *Proceedings of Interspeech*. 939–943.

[23] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio. 2019. The Pytorch-kaldi Speech Recognition Toolkit. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*. IEEE Press, Piscataway, NJ, USA, 6465–6469.

[24] Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of Conference on Natural Language Learning at HLT-NAACL*. ACL, Stroudsburg, PA, USA, 142–147.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 3104–3112.

[26] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[27] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence-video to text. In *Proceedings of International Conference on Computer Vision*. IEEE Press, Piscataway, NJ, USA, 4534–4542.

[28] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Michael Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2017. The microsoft 2016 conversational speech recognition system. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*. IEEE Press, Piscataway, NJ, USA, 5934–5938.

[29] Vikas Yadav and Steven Bethard. 2018. A Survey on Recent Advances in Named Entity Recognition from Deep Learning models. In *Proceedings of International Conference on Computational Linguistics*. ACL, Santa Fe, NM, USA, 2145–2158.

[30] Reza Yazdani, Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. The dark side of DNN pruning. In *Proceedings of Annual International Symposium on Computer Architecture*. ACM, New York, NY, USA, 790–801.

[31] Shiliang Zhang, Ming Lei, Zhijie Yan, and Lirong Dai. 2018. Deep-FSMN for Large Vocabulary Continuous Speech Recognition. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*. IEEE Press, Piscataway, NJ, USA, 5869–5873.

[32] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.

A PROOF OF LEMMA 1

Proof. For the sake of brevity we use $a_{i,j}$ to denote the transition probability from the sequence $\mathbf{y}_i$ in S_t to j-th state, given also the input sequence $\mathbf{x}^{|t-1|}$, i.e., $a_{i,j} = p(y_t = j|\mathbf{x}^{|t-1|}, \mathbf{y}_i)$. Recall that $\mathbf{y}_i$ is the i-th Sequence in S_{t-1}, as defined in Sec. 3.1.2, and by definition we have $\mathbf{s}_t(i,j) = (\mathbf{y}_i, y_t = j)$. We have:

$$
\begin{aligned}
H(\{\mathbf{s}_t(i,j)\}) &= -\sum_{i}^{|S_{t-1}|}\sum_{j}^{N} p(\mathbf{y}_i)a_{i,j}p_{t,j}\log(p(\mathbf{y}_i)a_{i,j}p_{t,j}) \\
&= -\sum_{i}^{|S_{t-1}|}(p(\mathbf{y}_i)\log p(\mathbf{y}_i)\sum_{j}^{N} a_{i,j}p_{t,j}) \\
&\quad -\sum_{j}^{N}(p_{t,j}\log p_{t,j}\sum_{i}^{|S_{t-1}|} p(\mathbf{y}_i)a_{i,j}) \\
&\quad -\sum_{i}^{|S_{t-1}|}\sum_{j}^{N} p(\mathbf{y}_i)a_{i,j}p_{t,j}\log a_{i,j}
\end{aligned}
$$

With $\sum_j^N a_{i,j}p_{t,j} \leq 1$ and $\sum_i^{|S_{t-1}|} p(\mathbf{y}_i)a_{i,j} \leq 1$, we find that

$$
\begin{aligned}
H(\{\mathbf{s}_t(i,j)\}) &\leq -\sum_{i}^{|S_{t-1}|} p(\mathbf{y}_i)\log p(\mathbf{y}_i) - \sum_{j}^{N} p_{t,j}\log p_{t,j} \qquad (3) \\
&\quad -\sum_{i}^{|S_{t-1}|}\sum_{j}^{N} a_{i,j}\log a_{i,j} \\
&= H(S_{t-1}) + H(\mathbf{p}_t) - \sum_{i}^{|S_{t-1}|}\sum_{j}^{N} a_{i,j}\log a_{i,j} \qquad (4)
\end{aligned}
$$

Since $H(S_{t-1})$ is fixed at time step t and $a_{i,j}$ is pre-learned, (4) is a strictly increasing function of $H(\mathbf{p}_t)$. □

B PROOF OF LEMMA 2

Proof. If $M = 1$ we have $H(\mathbf{v}) = 1$ as any other entropy $H(\mathbf{v})$ is infeasible. We find $B(\mathbf{v},\beta) = U_t(H(\mathbf{v}),\beta) = 1$ for all $0 \leq \beta \leq 1$.

At first let us suppose, that $\beta \leq 0.5$ and all vector elements satisfy $v_i \leq 0.5$ and therefore, the contributions to the entropy E are all monotonically increasing and concave. We show that a vector $\mathbf{v}$ which violates the bound cannot exist. Clearly, $B(\mathbf{v},\beta) > M$ or $B(\mathbf{v},\beta) > \lfloor 1/\beta \rfloor$ is impossible due to Lemma 1. Suppose there is a vector $\mathbf{v}$ with $B(\mathbf{v},\beta) > \lfloor H(\mathbf{v})/(\beta\log(1/\beta))\rfloor$. Then $\sum_{i=1}^{M} v_i\log(1/v_i) \geq B(\mathbf{v},\beta)\beta\log 1/\beta \geq (\lfloor H(\mathbf{v})/(\beta\log(1/\beta))\rfloor + 1)(\beta\log 1/\beta) > H(\mathbf{v})$.

Now let us look at the case where $v_1 > 0.5$. Note that obviously at most one element of $\mathbf{v}$ can exceed 0.5. Given v_1, we find that $0 \leq H(\mathbf{v}) \leq (1-v_1)\log((M-1)/(1-v_1)) + v_1 log(1/v_1)$ where the lower bound is obtained with $v_1 = 1$, and the upper bound reduces monotonically with increasing v_1 and is obtained with all other $M-1$ elements of p equal to $(1-v_1)/(M-1)$. Therefore, if $\beta > 0.5$ we find $U_t(H(\mathbf{v}),\beta) = 1$ if $H(\mathbf{v}) \leq (1-\beta)\log((M-1)/(1-\beta)) + \beta log(1/\beta)$ and $U_t(H(\mathbf{v}),\beta) = 0$ otherwise. If $\beta \leq 0.5$, then the bound $\lfloor H(\mathbf{v})/(\beta\log(1/\beta))\rfloor$ still holds as the overall achievable entropy $H(\mathbf{v})$ is smaller if one element satisfies $v_i > 0.5$.

For the tightness of the bound we restrict ourselves to $v_i \leq 0.5$ for brevity, We show that there exists a vector p for all terms of $B(\mathbf{v},\beta) = U_t(H(\mathbf{v}),\beta)$. Suppose $v_i = 1/M$ and $\beta \leq 1/M$, then $B(\mathbf{v},\beta) = M$; in this case, the two other terms of the upper bound are larger than M. Now suppose that $M_\beta = \lfloor 1/\beta \rfloor$ elements of p satisfy $v_i = \beta$, $\beta > 1/M$ and all other vector elements are $(1 - M_\beta\beta)/(M - M_\beta)$. One can show that in this case, the two other terms of the bound are larger than $\lfloor 1/\beta \rfloor$. Finally, suppose that $H(\mathbf{v}) = \log(M_\alpha)$ and $M_\alpha < M$ elements of p satisfy $v_i = 1/M_\alpha$ and all other elements are 0. If $\beta = 1/(M_\alpha + 1)$ we find $\lfloor H(\mathbf{v})/(\beta\log(1/\beta))\rfloor = \lfloor (M_\alpha+1)\log(M_\alpha)/\log(M_\alpha+1))\rfloor = M_\alpha$. Moreover, we have $\lfloor 1/\beta \rfloor = M_\alpha + 1$. □

C PROOF OF LEMMA 3

Proof. The probability distribution of $\mathbf{s}_t(i,j)$ is discrete and can be represented by some vector $\mathbf{v} \in \mathbb{R}^{|S_{t-1}|\times N}$ with $v_{i,j} = p(\mathbf{s}_t(i,j))$. And by definition we have $|S_t| = |\{(i,j)|p(\mathbf{s}_t(i,j)) \geq \beta\}| = B(\beta)$. Applying Lemma 2 we have that $B(H(\{\mathbf{s}_t(i,j)\},\beta)$ is monotonically increasing function of $H(\{\mathbf{s}_t(i,j)\})$. □