Learning of Discretized LSTMs

Nikolaus Kopp, Franz Pernkopf
SPSC Laboratory - Graz University of Technology
Christian Doppler Laboratory for Dependable Intelligent Systems in Harsh Environments
nkopp@tugraz.at, pernkopf@tugraz.at

The growing demand for both large-scale machine learning applications and AI models on embedded devices has created a need to miniaturize neural networks. A common approach is to discretize weights and activations, reducing memory footprint and computational cost. Many existing methods, however, rely on heuristic gradients or post-training quantization. Probabilistic approaches allow networks with discrete parameters and activations to be trained directly without such heuristics, yet their application to recurrent neural networks remains underexplored. In this work, we analyze several probabilistic training algorithms previously studied on feed-forward and convolutional networks, and demonstrate that the reparametrization trick can be effectively applied to LSTM networks with discrete weights. We investigate the effect of using step functions for individual LSTM gates, finding that binarizing the candidate and output gate can maintain performance, whereas binarizing the input gate severely degrades it. We show that probabilistic training pose a valuable alternative to quantization-aware training. Comparisons with continuous LSTMs paint a nuanced picture: in some cases, discrete valued networks match the results of continuous ones, while in others, discretization leads to a performance decline.

1. Introduction

Neural networks underpin many of today's most capable systems, from large-scale language models to medical and financial decision tools. While these models have achieved remarkable performance, they typically rely on energy-intensive data centers for both training and inference. In parallel, there is a growing demand for intelligent systems that operate on resource-constrained hardware such as mobile or embedded devices. These platforms cannot provide the same computational or energy budgets as large-scale infrastructure, motivating research into more efficient algorithms and hardware co-design. This growing demand has sparked renewed interest in methods that trade off accuracy, energy, and computational cost in a principled way.

A promising approach is to discretize the weights and activation functions of a network [1, 2], creating something we will refer to as *discrete network*. Binary or ternary networks dramatically reduce memory footprint and computational cost, as forward passes require far fewer operations. However, standard gradient-based training methods fail when weights or activations are discrete, since the gradients are either zero or undefined. To address this, Bengio et al. [3] have proposed a heuristic approach called the straight-through estimator (STE), which backpropagates gradients through discrete nodes using continuous approximations. Applying the STE concept to networks with discrete weights is called *quantization-aware training* (QAT). While effective in practice, these heuristics lack a rigorous theoretical foundation, making it difficult to guarantee convergence or to analyze their behavior formally.

Probabilistic training methods, including variational inference and the reparametrization trick [4], allow principled gradient updates even for discrete networks, and have been extended with techniques like the local reparametrization trick [5] and Gumbel-Softmax approximation [6]. However, applying these methods to recurrent networks introduces additional challenges. Recurrent connections involve shared weights across timesteps, long-term dependencies, and complex gradient interactions. However, probabilistic training of discrete recurrent networks remains largely unexplored, despite

Third Conference on Parsimony and Learning (CPAL 2026).

the growing interest in energy-efficient sequential models. This gap motivates our work: we systematically evaluate probabilistic and non-probabilistic strategies for training discrete long-short term memory (LSTM) networks and identify the design choices that most strongly impact performance. We further provide guidance for effectively training discrete LSTMs, reducing the memory footprint while minimizing loss in performance.

2. Related work

Resource-efficient deep learning methods Research has explored compressing and accelerating neural networks for deployment on resource-constrained devices. Popular techniques include pruning [7], knowledge distillation [8], and weight discretization [1, 9].

Heuristic gradients for discrete networks The straight-through estimator [3, 10] allows gradient propagation through discrete nodes using continuous approximations. This allows for QAT by keeping high-precision weights, applying discrete quantization during the forward pass and using surrogate gradients for the quantizers. Multiple variants of this approach have been proposed, including BinaryConnect [1] and WAGE [11], which rely on STE-based approximations to train networks with discrete weights, activations, or fully integer representations.

Probabilistic methods Probabilistic training treats weights as random variables and learns their posterior distributions. Two common approaches are: Markov Chain Monte Carlo [12], which approximates the posterior through sampling, and variational inference [13], where one learns a simplified posterior. Variational approaches were made popular for neural networks with the reparametrization trick [4], local reparametrization trick [5], and differentiable relaxations such as the Gumbel-Softmax trick [6, 14, 15]. Details will be presented below.

Probabilistic and discrete recurrent neural networks Training discrete recurrent neural networks (RNNs) using probabilistic methods remains largely unexplored. Fortunato et al. [16] investigated recurrent networks trained with variational inference, and Hajiramezanali et al. [17] extended similar ideas to graph neural networks. However, to our knowledge, no prior work has examined probabilistic training of discrete recurrent networks. A notable exception is Konrad [18], who analyzed the discretization of GRU gates using variational inference, however, they did not consider discrete weights.

3. Methodology

We consider supervised classification tasks, where we are given a dataset $\mathcal{D} = \{(\mathbf{x}_i, t_i)\}_{i=1}^N$ of input-target pairs and aim to model the relationship between inputs and targets. Formally, we are interested in the conditional distribution $p(t \mid \mathbf{x})$, which we approximate with a neural network $\hat{f}_{\mathcal{W}}(\mathbf{x})$ parametrized by weights $\mathcal{W}$. By defining a model $\hat{f}_{\mathcal{W}}(\cdot)$ and a loss function $\mathcal{L}(\mathcal{W}; \mathcal{D})$ that measures the discrepancy between the network predictions and the observed targets, we can turn supervised classification into an optimization problem: Finding the weights $\mathcal{W}$ that minimize this loss, we obtain a function $\hat{f}_{\mathcal{W}}(\cdot)$ which serves as an approximation to the unknown mapping.

In practice, neural networks are trained using gradient-based optimization: starting from random initial weights $\mathcal{W}$, these weights are iteratively updated using the gradient of the loss with respect to the weights, $\partial_{\mathcal{W}} \mathcal{L}(\mathcal{W}, \mathcal{D})$. However, such methods fail when the network contains discrete weights $\mathcal{W}$ or discrete activation functions, since the gradient is either zero or undefined, and thus provides no useful information for learning. Numerous methods have been proposed to address this issue, among which QAT has emerged as the most popular. QAT keeps continuous valued weights in the background which are discretized in the forward-pass, while in the backward-pass the non-differentiable quantizer is replaced by a surrogate partial derivative. We are specifically interested in binary $w \in \mathbb{D}_w = \{-1, 1\}$ and ternary weights $w \in \mathbb{D}_w = \{-1, 0, 1\}$, as well as using the binary activation functions. By binary activation function we mean either the sign or the step function. At such strong discretizations it is unclear whether the surrogate gradient provided by QAT allows

for a stable convergence to the minimum. To this end, we are interested in analyzing probabilistic training algorithms which provide a principled gradient.

3.1. Probabilistic description

Instead of selecting a single deterministic function $\hat{f}(\cdot) \in \{\hat{f}_{\mathcal{W}}\}_{\mathcal{W}}$, we aim to infer a distribution over functions $p(\hat{f}(\cdot))$ such that

$$\mathbb{E}_{\hat{f}\sim p(\hat{f}(\cdot))}[\hat{f}(\mathbf{x})] \approx p(t \mid \mathbf{x}).$$

Introducing the dataset $\mathcal{D}$, and treating the network parameters $\mathcal{W}$ as random variables, the predictive distribution can be written as

$$p(t \mid \mathbf{x}, \mathcal{D}) = \int_{\mathcal{W}} p(t \mid \mathbf{x}, \mathcal{W})\, p(\mathcal{W} \mid \mathcal{D})\, d\mathcal{W} = \mathbb{E}_{\mathcal{W}\sim p(\mathcal{W}\mid\mathcal{D})}\big[p(t \mid \mathbf{x}, \mathcal{W})\big], \tag{1}$$

provided that weights $\mathcal{W}$ are sufficient statistics. For simplicity, we present formulas for continuous weights; in the discrete case, the integral is replaced by a sum. Equation 1 provides an equivalent formulation of the expectation over functions. The term $p(t \mid \mathbf{x}, \mathcal{W})$ corresponds to the output of a network with parameters $\mathcal{W}$ and remains fixed during training, while $p(\mathcal{W} \mid \mathcal{D})$ is the posterior distribution determined from data.

3.2. Approximating the predictive distribution

Direct computation of Equation 1 is generally intractable. A standard approximation is Monte Carlo integration: sample $\mathcal{W}^{(s)} \sim p(\mathcal{W} \mid \mathcal{D})$ multiple times, compute their predictions and average them. By estimating with a single sample we get

$$\hat{f}_{\mathcal{W}^{(s)}}(\mathbf{x}) \approx \mathbb{E}_{\mathcal{W}\sim p(\mathcal{W}\mid\mathcal{D})}\big[p(t \mid \mathbf{x}, \mathcal{W})\big].$$

This connects deterministic network predictions to the probabilistic formulation. This means that when we know the posterior $p(\mathcal{W} \mid \mathcal{D})$ over categorically distributed weights, we can either sample from it or take the maximum a posteriori (MAP) estimate to obtain a deterministic discrete network. Thus, we now need to turn our attention to how we can obtain the posterior over categorically distributed weights.

3.3. The posterior $p(\mathcal{W} \mid \mathcal{D})$

By Bayes' theorem, the posterior can be expressed as

$$p(\mathcal{W} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathcal{W})\, p(\mathcal{W})}{p(\mathcal{D})}, \tag{2}$$

where $p(\mathcal{D} \mid \mathcal{W})$ is the likelihood, $p(\mathcal{W})$ the prior, and $p(\mathcal{D})$ the model evidence. The evidence is constant with respect to $\mathcal{W}$ and can be ignored. The prior encodes beliefs about weights before observing data, acting as a regularizer analogous to weight decay, and can enforce structural constraints. For instance, with ternary weights $\mathbb{D}_{\mathcal{W}} = \{-1, 0, 1\}$, the prior can encourage sparsity by favoring 0.

Assuming independent samples, ignoring $p(\mathbf{x})$, and modeling $p(t \mid \mathbf{x}, \mathcal{W})$ with a categorical distribution, the likelihood becomes

$$p(\mathcal{D} \mid \mathcal{W}) \propto \prod_{n=1}^{N} \prod_{c=1}^{C} \hat{f}_{\mathcal{W}}(\mathbf{x}_n)_c^{\mathbb{1}_{\{t_n=c\}}},$$

where $\mathbb{1}_{\{t_n=c\}}$ is the indicator function applied to target t_n and class index c, and $\hat{f}_{\mathcal{W}}(\mathbf{x}_n)_c$ denotes the c^{th} output of the network for sample n. Learning the posterior via Equation 2 is termed *Bayesian inference* and is intractable for multi-layer networks.

3.4. Variational inference

Variational inference (VI) is an approximation of Bayesian inference. In VI we fix the posterior to a simplified structure $q(\mathcal{W} \mid \boldsymbol{\nu})$, where $\boldsymbol{\nu}$ are the variational parameters that define this distribution. We adopt the mean-field assumption that all weights are mutually independent. For continuous weights we use the Gaussian distribution

$$q(\mathcal{W} \mid \boldsymbol{\nu}) = \prod_{w \in \mathcal{W}} \mathcal{N}(w \mid \boldsymbol{\nu}_w) = \prod_{w \in \mathcal{W}} \mathcal{N}(w \mid \mu_w, \sigma_w^2),$$

where $\boldsymbol{\nu}_w = \{\mu_w, \log \sigma_w^2\}$ are the mean and the log variance parameters. We model discrete weights $w \in \mathcal{W}$ using a categorical distribution,

$$q(\mathcal{W} \mid \boldsymbol{\nu}) = \prod_{w \in \mathcal{W}} \mathrm{Cat}(\mathbf{p}_w \mid \boldsymbol{\nu}_w),$$

where $\boldsymbol{\nu}_w$ are the unnormalized logits that parameterize the probability masses $\mathbf{p}_w = \mathrm{softmax}(\boldsymbol{\nu}_w)$.

We aim to learn the approximate posterior $q(\mathcal{W} \mid \boldsymbol{\nu})$ by adjusting $\boldsymbol{\nu}$ such that it is as close as possible to the true posterior $p(\mathcal{W} \mid \mathcal{D})$. This is achieved by minimizing the Kullback–Leibler divergence between them, leading to the loss function known as the *variational free energy* (VFE):

$$\mathcal{F}(\boldsymbol{\nu}, \mathcal{D}) = \underbrace{\mathrm{KL}(q(\mathcal{W} \mid \boldsymbol{\nu}) \,\|\, p(\mathcal{W}))}_{\mathcal{L}_C} + \underbrace{\mathbb{E}_{\mathcal{W} \sim q(\mathcal{W} \mid \boldsymbol{\nu})}[-\log p(\mathcal{D} \mid \mathcal{W})]}_{\mathcal{L}_E}.$$

We refer to $\mathcal{L}_C$ as the *complexity loss* and to $\mathcal{L}_E$ as the *expected negative log-likelihood*. Analogous to deterministic learning, we iteratively update the parameters $\boldsymbol{\nu}$ using the gradient $\nabla_{\boldsymbol{\nu}} \mathcal{F}(\boldsymbol{\nu}, \mathcal{D})$. While $\nabla_{\boldsymbol{\nu}} \mathcal{L}_C$ can be computed analytically, the gradient of the expected term $\nabla_{\boldsymbol{\nu}} \mathcal{L}_E$ must be estimated.

3.5. Estimating the gradient of the expected negative log-likelihood

There exist several approaches to estimate $\nabla_{\boldsymbol{\nu}} \mathcal{L}_E$. In practice most methods rewrite the gradient of an expectation as an expectation of a gradient and approximate this expectation by Monte Carlo integration. Examples of unbiased but high-variance estimators are REINFORCE [19] and the ARM algorithm [20]; their variance typically impedes convergence in neural networks, rendering them impractical in many settings.

We focus on the reparametrization- [4] and the local reparametrization trick [5], which we refer to as r-trick and lr-trick, respectively. The r-trick relies on rewriting the sampling operation $\mathcal{W} \sim q(\mathcal{W} \mid \boldsymbol{\nu})$ as a deterministic transformation

$$\mathcal{W} = \tilde{g}(\boldsymbol{\nu}, \boldsymbol{\epsilon}),$$

with $\boldsymbol{\epsilon}$ drawn from a fixed base distribution. We call $\tilde{g}(\cdot, \cdot)$ a reparametrization. This permits pushing the gradient inside the expectation:

$$\nabla_{\boldsymbol{\nu}} \mathbb{E}_{\mathcal{W} \sim q(\mathcal{W} \mid \boldsymbol{\nu})}\big[-\log p(\mathcal{D} \mid \mathcal{W})\big] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[-\frac{\partial \log p(\mathcal{D} \mid \tilde{g}(\boldsymbol{\nu}, \boldsymbol{\epsilon}))}{\partial \tilde{g}(\boldsymbol{\nu}, \boldsymbol{\epsilon})} \frac{\partial \tilde{g}(\boldsymbol{\nu}, \boldsymbol{\epsilon})}{\partial \boldsymbol{\nu}}\right]. \tag{3}$$

For discrete random variables we employ the Gumbel–Softmax relaxation [6], a differentiable but biased reparameterization of the categorical distribution. The STE can be combined with this relaxation to perform exact discrete sampling in the forward pass while using the continuous surrogate in the backward pass. The Gumbel-Softmax trick approximates the argmax function with the softmax, which comes with a hyperparameter termed *temperature* τ. Through the temperature τ we can control the bias–variance trade-off: as $\tau \to 0$ the relaxation approaches the true discrete sample but the variance goes to ∞ (bias $\downarrow$, variance $\uparrow$), whereas as $\tau \to \infty$ the relaxation becomes overly smooth (variance $\downarrow$, bias $\uparrow$).

The lr-trick pushes the same idea to activations by expressing their distribution as a transformation of random weights and sampling the activations directly via reparameterization. When weights are Gaussian, the induced activations are also Gaussian and their moments can be computed analytically.

Shayer et al. [21] generalized this argument to discrete weights using the generalized Central Limit Theorem, which motivates approximating the sum of many independent discrete variables by a Gaussian. They empirically showed that the Gaussian approximation is often weak but can be improved by penalizing low weight entropy.

A key limitation of the lr-trick is the assumption that summands are statistically independent. Weight sharing (on which recurrent networks rely) violates this assumption and introduces additional correlations that bias the gradient estimate. Crucially, the two sources of bias — discreteness and shared weights — depend on weight entropy in opposite ways: increasing entropy tends to reduce the bias arising from discrete (non-Gaussian) weights, but it can amplify the bias induced by shared weights. Consequently, for discrete recurrent networks the lr-trick exhibits a compound, hard-to-characterize bias; entropy-penalization can partially mitigate one source of bias but may worsen the other, so applying the lr-trick in such models requires caution. The lr-trick has substantially lower variance than the r-trick, however, due to the lack of control over the bias in the setting of discrete recurrent neural networks, the lr-trick loses it's dominance over the r-trick.

3.6. Scaling activations

When using binary or ternary weights, the resulting activations can become large enough to saturate the logistic sigmoid or hyperbolic tangent functions. To stabilize training for QAT, r-trick and lr-trick, we introduce a scaling factor for each gate, which scales the activations. These scaling factors are jointly learned with the network weights. Note, that this scaling is mathematically equivalent to scaling the corresponding binary or ternary weight values [22].

4. Datasets

We evaluate our probabilistic methods for discrete LSTMs on four benchmark datasets: the MNIST pen-stroke digits (MNIST) dataset [23], the Australian Sign Language (Auslan) dataset [24], the UCI Human Activity Recognition (UCIHAR) dataset [25], and the Epileptic Seizure Recognition (Epileptic) dataset [26].

For MNIST, we subsample 10,000 training, 2,000 validation, and 2,000 test examples to reduce computational cost while preserving the dataset's diversity. In the Auslan dataset, sequences longer than 70 time steps are discarded, as only a small fraction exceed this length; this reduces unnecessary padding and limits the maximum sequence length for efficient processing. All datasets are normalized to have zero mean and unit variance along each feature dimension.

5. Experiments

All experiments are conducted using networks with two LSTM layers followed by a fully connected output layer. We use an adaptation of the LSTM cell that combines the input and forget gates as $f = 1 - i$. The number of hidden units per layer was determined via architecture search on a continuous LSTM baseline. Details are in Appendix A. Discretizing these networks yields low-complexity models, and therefore additional regularization did not provide improvements. To this end, we omit the complexity loss $\mathcal{L}_C$ in our experiments. Weights $\mathcal{W}$ and distribution parameters $\boldsymbol{\nu}$ are updated using the Adam optimizer. All biases remain continuous. Categorical parameters are initialized using a Dirichlet process, where each component of the concentration parameter is set to 1.

We evaluate network performance using two discretization-based metrics. In the *sample* metric, each network is sampled five times and their mean accuracy is reported, whereas in the *MAP* metric, the maximum a posteriori of the network is used. We employ post-training early stopping: the networks are trained sufficiently long, and the test set metric is reported at the epoch corresponding to the highest value of the respective validation metric. Each experiment is repeated over three independent trials.

5.1. LSTM gate discretization

To systematically assess the effects of gate discretization in LSTMs, we first conduct an ablation study using QAT. We consider ternary weights and discretize different subsets of gate activations within each LSTM cell at varying discretization levels (2, 3, 4, and 8). All other activations remain continuous. The resulting performance is displayed in Figure 1.

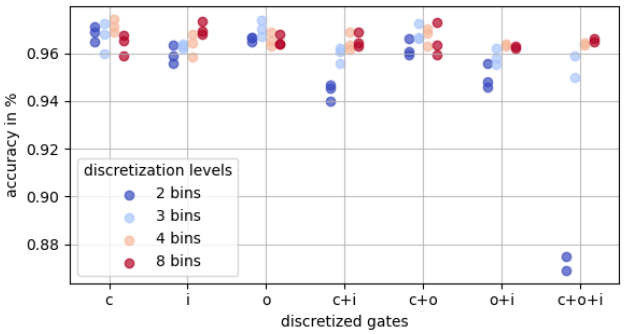

Figure 1: Accuracies on MNIST test set of LSTM networks with ternary weights when discretizing different subsets of gate activations. The x-axis shows which gates were discretized, where c is the input-, o the output- and c the candidate gate. For example, *o+i* means the output- and input gate were discretized. We further varied the discretization level, from just 2 levels (binary activation function) up to 8.

The experiment reveals a hierarchy in gate importance. Discretizing the candidate and output gate results in the smallest degradation in performance, whereas discretizing the input gate leads to the largest. When all gates are discretized using binary gates, the performance drop becomes substantial. Consequently, for a final comparison we restrict gate binarization to the candidate and output gates while keeping a continuous input gate activation, as this configuration minimizes the loss in accuracy. Notably, the adverse effects of discretization can be mitigated by introducing multiple discretization levels.

5.2. Analyzing temperature τ in Gumbel-Softmax

We next conduct an ablation study on MNIST to assess the interaction between the reparametrization trick (r-trick), the STE, and the Gumbel-Softmax temperature τ. Both binary $w \in \mathbb{D}_w = \{-1, 1\}$ and ternary $w \in \mathbb{D}_w = \{-1, 0, 1\}$ weights are considered. For each weight type, we vary τ to analyze its effect on convergence with and without STE. Recall, that STE means that we do exact sampling in the forward-pass and use Gumbel-Softmax only in backward-pass. The networks are discretized using the MAP. Figure 2 shows the resulting accuracies on the test set.

Recall that a low temperature τ yields gradient estimates with low bias but high variance. While high variance affects convergence speed, it should not degrade the final performance. Because our optimizer employs moment-based updates, the variance primarily reduces the update magnitude rather than altering its direction. Consequently, the results align with our expectation that LSTMs can achieve high accuracy even at low temperature τ.

In contrast, when no STE is used, a high temperature τ introduces substantial bias in the forward pass, causing gradients to be evaluated at incorrect points and hindering convergence. With STE, the bias appears only during backpropagation and affects just the partial derivative of the corresponding weight, making the estimator markedly more robust to temperature variations.

Moreover, LSTM performance appears more sensitive to τ when binary weights are used, a pattern observed both with and without the STE. Overall, the experiment indicates that the STE should be preferred: it exhibits no observable downside and provides strong robustness to τ. In all subsequent experiments, we therefore set $\tau = 1$ and employ Gumbel–Softmax in combination with STE.

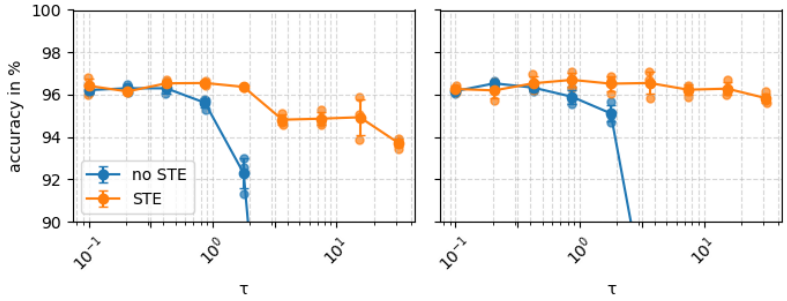

Figure 2: Accuracy on MNIST test set of discrete LSTM networks as a function of Gumbel-Softmax temperature τ. The networks were trained with reparametrization trick, both with and without STE, and then discretized using the MAP weights. Note, that *without STE* means that we apply the Gumbel-Softmax trick in the forward- and backward-pass. In contrast, *with STE* means that we do exact sampling in the forward-pass and only apply Gumbel-Softmax trick in the backward-pass. **Left**: Binary weights. **Right**: Ternary weights.

5.3. Weight entropy

We next examine how the entropy of categorical weight distributions evolves during training on the MNIST dataset. To this end, we train LSTMs with binary and ternary weights using both the r-trick (with $\tau = 1$ and an STE) and the lr-trick, and evaluate the entropy every 10 epochs. Figure 3 shows how the average weight entropy changes over training and how it relates to the performance of the discretized LSTMs. For networks with ternary weights, we additionally evaluated the average probability mass across the three weight values and found that each value remained approximately equally likely.

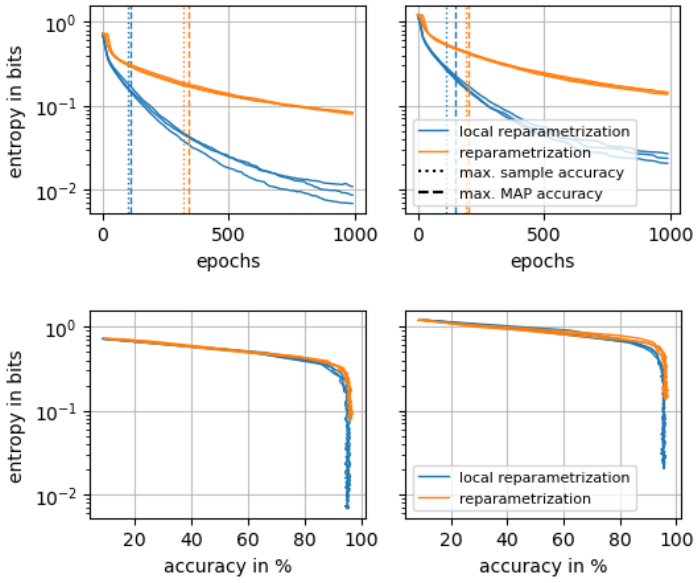

Figure 3: Average entropy of categorical weights plotted over the training epochs (top) and over the performance of sampled networks (bottom) on the validation set for the r-trick and lr-trick. The vertical lines in the top plots show the average epoch (over three trials) at which the accuracy of the discretized network on the validation set came within 99% of it's maximum value. This discretization was done through both sampling and setting weights to MAP. **Left**: Binary weights. **Right**: Ternary weights.

Without regularization, the weight distributions tend to become deterministic, with no particular weight value favored. Networks trained with the r-trick, however, converge at higher weight entropies than those trained with the lr-trick. This behavior is consistent across performance metrics—whether measured on training or validation data, and whether the network is discretized or not. The apparent linear relationship at low accuracies in the lower plots of Figure 3 may be an artifact of having only few data points in that region.

5.4. Benchmark Comparison Across Discretization Levels

In this set of experiments, we compare the performance of discrete LSTM networks trained with probabilistic and deterministic approaches against continuous-valued LSTM networks with same architecture. Specifically, we evaluate QAT, the reparametrization trick (r-trick), and the local reparametrization trick (lr-trick), alongside conventional backpropagation.

We consider three levels of discretization: (i) ternary weights $w \in \{-1, 0, 1\}$, (ii) binary weights $w \in \{-1, 1\}$, both with conventional gate activation functions, and (iii) ternary weights with binary candidate and output gates and continuous input gate.

Table 1: Comparison of accuracies across all datasets for different discretization levels and algorithms. The column *binary gate* shows which gate activations were binarized; if not, standard nonlinearities are used. The number of bits used to store weights and biases is shown for the respective discretization level. Deterministic learning refers to standard backpropagation and QAT for continuous- and discrete valued weights, respectively. For r-trick and lr-trick, performance is reported for both sampling and MAP discretization.

Dataset	Weight type	Binary gate			Bits	Deterministic	r-trick		lr-trick	
		c	o	i			Sample	MAP	Sample	MAP
MNIST	continuous	✗	✗	✗	732480	96.13 ± 0.31	-	-	-	-
	ternary	✗	✗	✗	54720	96.43 ± 0.24	96.72 ± 0.14	96.60 ± 0.62	96.40 ± 0.12	96.60 ± 0.16
	binary	✗	✗	✗	32128	96.73 ± 0.22	96.38 ± 0.10	96.47 ± 0.21	96.32 ± 0.06	96.40 ± 0.07
	ternary	✓	✓	✗	54720	96.37 ± 0.19	96.40 ± 0.10	96.78 ± 0.08	95.90 ± 0.05	95.77 ± 0.37
	binary	✓	✓	✗	32128	95.50 ± 0.45	95.93 ± 0.61	96.05 ± 0.31	95.57 ± 0.52	95.52 ± 0.21
Auslan	continuous	✗	✗	✗	933888	94.83 ± 0.16	-	-	-	-
	ternary	✗	✗	✗	69888	94.88 ± 0.17	95.27 ± 0.87	95.71 ± 1.23	93.95 ± 1.56	92.52 ± 1.24
	binary	✗	✗	✗	41088	94.28 ± 0.68	94.17 ± 1.95	90.98 ± 1.27	90.98 ± 2.18	87.13 ± 3.52
	ternary	✓	✓	✗	69888	97.36 ± 0.47	97.25 ± 0.82	92.85 ± 1.73	93.29 ± 0.56	91.86 ± 1.53
	binary	✓	✓	✗	41088	91.42 ± 0.93	94.94 ± 0.95	88.67 ± 4.21	91.75 ± 0.71	90.10 ± 1.23
UCIHAR	continuous	✗	✗	✗	2894016	93.51 ± 0.75	-	-	-	-
	ternary	✗	✗	✗	198336	89.89 ± 2.16	92.79 ± 0.26	92.06 ± 0.57	91.65 ± 1.25	91.19 ± 1.30
	binary	✗	✗	✗	108480	91.64 ± 1.02	92.17 ± 1.08	89.23 ± 2.08	92.75 ± 0.50	91.49 ± 0.20
	ternary	✓	✓	✗	198336	91.21 ± 0.57	90.35 ± 0.23	88.10 ± 1.48	90.05 ± 0.32	88.90 ± 1.29
	binary	✓	✓	✗	108480	90.62 ± 0.30	82.57 ± 5.75	80.16 ± 5.09	90.65 ± 1.14	88.82 ± 0.42
Epileptic	continuous	✗	✗	✗	177813	72.03 ± 0.65	-	-	-	-
	ternary	✗	✗	✗	11253	74.13 ± 0.26	73.26 ± 1.58	72.64 ± 1.44	69.87 ± 1.80	70.97 ± 0.31
	binary	✗	✗	✗	5701	72.19 ± 0.18	74.13 ± 0.74	72.09 ± 0.65	72.36 ± 1.48	72.01 ± 1.97
	ternary	✓	✓	✗	11253	70.30 ± 1.36	71.84 ± 0.72	71.25 ± 0.67	69.62 ± 0.42	70.01 ± 0.23
	binary	✓	✓	✗	5701	67.52 ± 0.92	70.51 ± 0.25	68.41 ± 0.62	68.50 ± 0.11	67.70 ± 0.48

The results are shown in Table 1. We refrain from highlighting the best accuracies because there are no consistent winners and because training discrete LSTMs is somewhat unstable: certain trials converge to suboptimal solutions, lowering the mean accuracy. This effect is visible in Table 1, where a large standard deviation indicates poor convergence of a single trial. However, such outliers do not render the training algorithms ineffective. Repeating training multiple times allows to identify these weak trials through their poor validation performance.

A consistent trend across datasets is that MAP discretization rarely outperforms sampling based discretization, and in several cases performs substantially worse. This suggests that sampling is generally superior, implying that probability mass around the mode carries useful information.

Recall that both the r-trick and the lr-trick tend to reduce weight entropy as training progresses. As the distributions become increasingly deterministic, MAP and sample discretizations coincide. Therefore, it may be beneficial to prevent excessive entropy collapse, for instance by penalizing low entropy.

Moreover, the r-trick tends to outperform the lr-trick. We hypothesize that this difference arises from biases introduced by simultaneously using discrete weights and shared weights across time. In prior work, the lr-trick is often favored due to its faster convergence; hence, it is noteworthy that it performs poorly for discrete LSTMs. Nevertheless, it remains useful for pretraining such networks.

Although definitive conclusions between QAT and r-trick are difficult, the results suggest that the r-trick (sampled networks) may achieve slightly higher accuracy than QAT. More extensive experiments would be required to assess the statistical significance of this observation; therefore, we refrain from claiming that probabilistic training is superior to QAT. We also want to add that an additional feature of the r-trick is that its bias and variance can be controlled via the temperature parameter τ. Early in training, higher temperatures may allow for faster convergence despite less accurate gradients, while later epochs could benefit from lower temperatures for more precise gradient estimates. This behavior could be achieved by annealing τ over the course of training, although we did not explore this in the present study.

The main goal of discretization is to reduce memory footprint while maintaining performance. While our results indicate that this is achievable for certain datasets and discrete configurations, especially when we do not binarize gates, we did not perform comparisons on smaller architectures. While we aimed to identify the smallest architecture on which the continuous network reaches maximal performance, the search was not exhaustive. On the Auslan dataset we can observe that the chosen architecture was indeed not optimal: the LSTMs with ternary weights and binary gates substantially outperform the continuous baseline. This suggests that the continuous architecture had too high complexity and that the discrete setting provided useful regularization. Thus, a systematic ablation study comparing continuous and discrete networks across a range of smaller to larger architectures would provide deeper insight.

6. Limitations

Our study has several limitations. First, the ablation studies were conducted on a single dataset, which may limit the generality of our findings. Second, we focused exclusively on pure LSTMs, which are often outperformed by modern sequence modeling. Third, we did not perform systematic comparisons on smaller architectures, leaving open questions about miniaturization. Finally, we did not evaluate our methods across a broad range of datasets.

7. Conclusion

We investigated deterministic and probabilistic training strategies for obtaining discrete LSTMs, focusing on binary and ternary weights and binary activation functions. Our experiments demonstrate that training discrete LSTMs is feasible and that probabilistic algorithms constitute a viable alternative to the widely used QAT. We further show that the impact of binarizing gate activation functions strongly depends on which gate is discretized: LSTM performance critically relies on a continuous-valued input gate, but allowing multiple discretization levels can prevent a drop in performance. Overall, the r-trick outperforms the lr-trick for discrete LSTMs, and sampling from the posterior generally yields better results than setting weights to the MAP. Additionally, we demonstrate that Gumbel-Softmax should be used in conjunction with STE.

As potential avenues for future research, temperature annealing and entropy regularization could be explored to analyze their effects and potentially improve training stability and convergence. Further work may also investigate how performance scales with network size and extend these methods beyond pure LSTMs to architectures such as CNN-LSTMs or attention-based models.

Acknowledgments and Disclosure of Funding

The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. Furthermore, this research was partially funded by the Austrian Science Fund (FWF) **10.55776/PAT7753623**.

References

[1] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[2] Wolfgang Roth, Günther Schindler, Holger Fröning, and Franz Pernkopf. Training discrete-valued neural networks with sign activations using weight distributions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 382–398. Springer, 2019.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[5] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[6] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[7] Jinyang Guo, Wanli Ouyang, and Dong Xu. Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1508–1517, 2020.

[8] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.

[9] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[11] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[12] Rohitash Chandra and Joshua Simmons. Bayesian neural networks via mcmc: a python-based tutorial. *IEEE Access*, 12:70519–70549, 2024.

[13] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

[14] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[15] Wolfgang Roth and Franz Pernkopf. Discrete-valued neural networks using variational inference. 2018.

[16] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017.

[17] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

[18] Alexander Konrad. Learning discrete gates for grus with variational inference. Master's thesis, University of California, Irvine, 2022.

[19] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[20] Mingzhang Yin and Mingyuan Zhou. Arm: Augment-reinforce-merge gradient for stochastic binary networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[21] Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. In *International Conference on Learning Representations (ICLR)*, 2018.

[22] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017.

[23] Edwin D. de Jong. MNIST digits as stroke sequences [code repository]. GitHub, 2016.

[24] Mohammed Kadous. Australian Sign Language Signs (High Quality). UCI Machine Learning Repository, 2002. DOI: https://doi.org/10.24432/C5VG6R.

[25] Jorge Reyes-Ortiz, Davide Anguita, Alessandro Ghio, Luca Oneto, and Xavier Parra. Human Activity Recognition Using Smartphones. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C54S4K.

[26] Q. Wu, E. Fokoue, and R. G. A. Epileptic seizure recognition data set. UCI Machine Learning Repository, 2017.

A. Appendix

Details of datasets and LSTM architectures are displayed in Table 2.

Table 2: Datasets and used LSTM architectures. Shown are the number of samples in training set $|\mathcal{D}_{tr}|$, feature dimensionality, number of classes, the average sequence length T_{avg} in training set, and the layout of the LSTM network. The layout is shown as $l_1 - l_2$, where l_1 and l_2 corresponds to the number of units in first and second LSTM layer, respectively. The third layer is always a fully-connected feed-forward layer, whose dimension depends on the number of classes.

Dataset	$\lvert\mathcal{D}_{tr}\rvert$	**Features**	**Classes**	T_{avg}	**Layout**
MNIST	10000	4	10	38.99	64-32
Auslan	1900	22	96	55.87	64-32
UCIHAR	5881	9	6	128	128-64
Epileptic	7360	1	5	178	32-16