Quantizing Spiking Neural Networks with Integers

Clemens JS Schaefer and Siddharth Joshi
{cschaef6,sjoshi}2@nd.edu
University of Notre Dame
Notre Dame, Indiana

ABSTRACT

Spiking neural networks (SNNs) are a promising approach to developing autonomous agents that continuously adapt to their environment. Developing low-power SNNs that can be implemented on a digital platform is a critical step to the realization of such agents. One of the most important methods of implementing low-power SNNs requires operating at reduced precision. While traditional computer vision has seen a lot of research examining the trade-offs between precision and model performance, such trade-offs are underexamined for contemporary SNNs. This paper studies the trade-offs associated with learning-performance and the quantization of neural dynamics, weights and learning components in SNNs. Our results show that SNNs trained using only integer fixed-point representations can still retain their accuracy while occupying dramatically lower memory footprints and using only energy-efficient fixed-point arithmetic. We show that the memory usage of SNNs trained with reduced precision weights, errors, gradients and neural dynamics can be downsized by 73.78% at the cost of 1.04% test error increase on the DVS gesture data set.

KEYWORDS

Spiking Neural Networks, Quantization, Memory Efficiency, Local Learning, Quantized Learning, Surrogate Gradient

ACM Reference Format:
Clemens JS Schaefer and Siddharth Joshi. 2020. Quantizing Spiking Neural Networks with Integers. In *International Conference on Neuromorphic Systems 2020 (ICONS 2020), July 28–30, 2020, Oak Ridge, TN, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3407197.3407203

1 INTRODUCTION

Developing mobile artificial autonomous agents that are able to learn from and adapt to their environment remains a grand challenge in engineering. Since mobility is key to such systems, any algorithms used to endow adaptivity/intelligence to such systems must operate under stringent power-performance constraints [3]. Recently developed autonomous agents have adopted artificial neural network (ANN) based solutions, in part due to their superior pattern recognition performance, when trained using abundant data [14]. This remarkable performance has also been replicated in spiking neural networks (SNNs), often while incurring lower computational costs [23, 24, 27]. This makes SNNs, particularly attractive in the context of developing mobile, autonomous agents [8, 12, 19].

SNNs encode information using a unary, temporal code which is used for both computation and communication [9, 17], with the temporal dynamics playing an important role in efficiently processing information that evolves over time [21, 26]. Building on that recent work shows continuous, supervised learning in such networks [13]. Developing application-specific digital systems for efficiently implementing these networks for continuous learning requires: 1) embedding primitives within the hardware that enable learning, 2) quantized operation, for energy-efficiency in the mobile setting, and 3) hardware primitives to efficiently emulate temporal dynamics required for SNN operation. While, individually, each one of these properties can, and have, been shown in prior work [2, 18], their interaction and trade-offs in the context of SNNs requires more study. For example, while ANNs have been trained at low-precision [30] and SNNs have been fine-tuned for low-precision inference after high-precision training [18], the effects of restricting the temporal dynamics in SNNs to forms that are amenable to efficient-digital hardware are still active areas of investigation [32].

This paper studies the trade-offs associated with learning performance and quantization in SNNs, focusing on the deep continuous local learning (DECOLLE) SNN architecture introduced in [13]. We first introduce some necessary background and terminology in Section 2, followed by an analysis of how we implement integer quantization of the various SNN parameters in Section 3 and its impact on learning performance in Section 4. We summarize and conclude the paper in Section 5.

2 BACKGROUND AND RELATED WORK

Historically, the design of SNNs has been intertwined with the design of power-efficient hardware platforms [7]. Indeed, the appeal of SNNs has always been tightly coupled with arguments relating to the energy-efficiency of the biological brain [32]. Thus, SNNs must be designed to be computational efficiency when implemented in hardware. Indeed, multiple researchers have examined the number of synaptic operations an SNN might incur and contrasted them with those of more traditional neural networks (NNs) [23, 27], finding that SNNs were able to achieve the same level of accuracy while incurring fewer operations/having lower activity. There have also been studies on operating SNNs composed of reduced precision weights (9-bit [31] and 1-bit [32]) combined with spike-timing dependent plasticity (STDP) learning rules.

More recent SNN models step away from STDP, adopting gradient based learning rules [11, 21, 26, 33] instead. However, all of these models rely on the temporal evolution of neuron/synapse dynamics in some form or the other. Thus, if such SNNs are to be implemented on custom digital hardware, their performance when operating on highly quantized systems must be examined critically.

ICONS 2020, July 28–30, 2020, Oak Ridge, TN, USA

ACM ISBN 978-1-4503-8851-1/20/07…$15.00
https://doi.org/10.1145/3407197.3407203

2.1 Spiking Neural Networks

We follow the neuron model outlined in [13, 22] and provide a brief introduction to that leaky integrate-and-fire model here. When operating over discrete time-steps (n), the leaky integrate-and-fire (LIF) neuron model [4, 13] can be expressed as:

$$U_i^{(l)}[n] = \sum_j W_{ij}^{(l)} P_j[n] - \delta R_i[n], \tag{1}$$

$$S_i^{(l)}[n] = \Theta(U_i^{(l)}[n] - \vartheta), \tag{2}$$

$$Q_j[n+1] = \alpha Q_j[n] + S_j^{(l-1)}[n],$$
$$P_j[n+1] = \beta P_j[n] + Q_j[n],$$
$$R_i[n+1] = \gamma R_i[n] + S_i^{(l)}[n].$$

The membrane potential of neuron i in layer l at time-step n is denoted by $U_i^{(l)}[n]$. A spike ($S_i^{(l)}[n]$) is issued at time-step n, once the membrane potential equals or exceeds the threshold ϑ. This is expressed through the step function Θ in (2). W_{ij} represents the synaptic weight between pre-synaptic neuron j and post-synaptic neuron i, meanwhile α, β, and γ are decay constants for the membrane potential, synapse state, and refractory state respectively. The variable $Q_j[n]$ represents the eligibility trace of the current-based synapse to neuron j in the previous layer at time step n. $P_j[n]$ is the membrane trace corresponding to $Q_j[n]$. The LIF reset is governed by $R_i[n]$ and the magnitude of the reset is controlled by δ, see Eq. 1.

2.2 Training SNNs with Deep Continuous Local Learning

For this work, we examine the impact of quantization on training SNN models based on the DECOLLE architecture [13, 22]. This architecture uses surrogate gradients and a three-factor learning rule from [33] together with local learning [20]. Given a loss function based on output spikes ($S_i^{(l)}$), applying standard gradient descent and the chain rule yields an expression like (3) for the weight gradients

$$\frac{\partial}{\partial W_{ij}}\mathcal{L} = \frac{\partial}{\partial S_i}\mathcal{L}\frac{\partial}{\partial U_i}S_i\frac{\partial}{\partial W_{ij}}U_i. \tag{3}$$

In the DECOLLE framework, the $R_i[n]$ term in U_i from Eq. 2 is ignored in the gradient computation, yielding $\frac{\partial}{\partial W_{ij}}U_i = P_j[n]$. Further, since the threshold Θ induces a discontinuity, DECOLLE employs a surrogate gradient (the derivative of a sigmoid function) based on the membrane potential [13] to overcome this discontinuity. Thus, $\frac{\partial}{\partial U_i}S_i$ becomes $\sigma'(U_i^l[n])$. Finally, the error term, $\frac{\partial}{\partial W_{ij}}U_i$, is required for credit assignment and rather than using traditional back-propagation, DECOLLE uses a local classifier to provide an error ($E_i[n]$) for each layer. Taken together, the weight update equation can now be written as:

$$-\Delta W_{ij}^l \propto \frac{\partial}{\partial W_{ij}}\mathcal{L} = P_j[n]\sigma'(U_i^l[n])E_i[n].$$

This expression is easily evaluated and each variable represents contributions from different terms in the equation (1). The first factor is $P_j[n]$, contributing a signal from the pre-synaptic potential, the second factor is an expression containing $U_i[n]$, the membrane potential, giving post-synaptic information. The last term is an external error at time-step n, given as $E_i[n]$.

Our choice in SNN architecture is motivated by the implementation of credit assignment, alternative gradient based SNNs [26, 33] are much more computationally intensive, requiring backpropagation-through-time (BPTT) [29]. However, DECOLLE uses a local learning, rule derived from [20], making its implementation more suited to low-power, autonomous systems.

We generate our error term through a fixed local random local classifier, fed with the membrane potential of neurons, thereby avoiding the spiking discontinuity. The membrane is processed through a sigmoid for the local classifier. Allowing for training using a surrogate gradient based on the derivative of the sigmoid function. For this fixed classifier, we ensure that the weights of the backward pass are sign concordant with the weights on the forward pass [16].

3 QUANTIZING SNNS

Due to the hardware benefits offered by low-precision operations [32], some SNN hardware platforms do implement integer arithmetic [10] and quantized storage and operation [18]. Indeed, the benefits to operating on heavily quantized values can readily be seen through the following observations: 1) operating on and storing weights with fewer bits, reduces the memory footprint of SNNs consequently reducing power, 2) digital circuits that implement arithmetic on quantized integers are dramatically simpler, and thus more energy-efficient. The authors in [32] estimate an order of magnitude reduction in power that can be gained from SNNs using low-bit synaptic weights during inference. These savings are magnified when SNNs learn using fewer bits for computation and communication.

Figure 1 diagrams the quantization scheme we followed in this paper. As we described in Section 2, the SNN architecture is split into a segment that encodes everything using spike-timing (shown in blue in Figure 1) and a segment that encodes information using rates (shown in green in Fig. 1). The signal flow graph in Fig. 1, shows how data flows in the network along with an overview of our scheme to emulate quantized hardware operation in software. Neural and synaptic dynamics are implemented as simple first-order, low-pass, IIR filters, with gain-coefficients α, β, and γ.

3.1 Quantizing the Local Classifier

Since the local classifier contains no temporal dynamics, we use existing state-of-the-art integer quantization methods that are applied to ANN quantization [30]. This involves two steps: i) quantization through integer rounding and ii) clipping the weights into a range determined by the number of bits (k). Lets denote quantization by a function $\zeta(x, k)$ which takes a floating-point variable x and quantizes it into a k-bit fixed-point representation as:

$$\zeta(x,k) = \text{clip}\left\{\omega(k)\text{round}\left[\frac{x}{\omega(k)}\right],\ -1+\omega(k),\ +1-\omega(k)\right\}, \tag{4}$$
$$\omega(k) = 2^{1-k}.$$

The quantized values are treated like fixed-point fractions between +1 and −1, with one bit dedicated to storing sign information. As shown in (5), the weights of the local classifier (S) are initialized

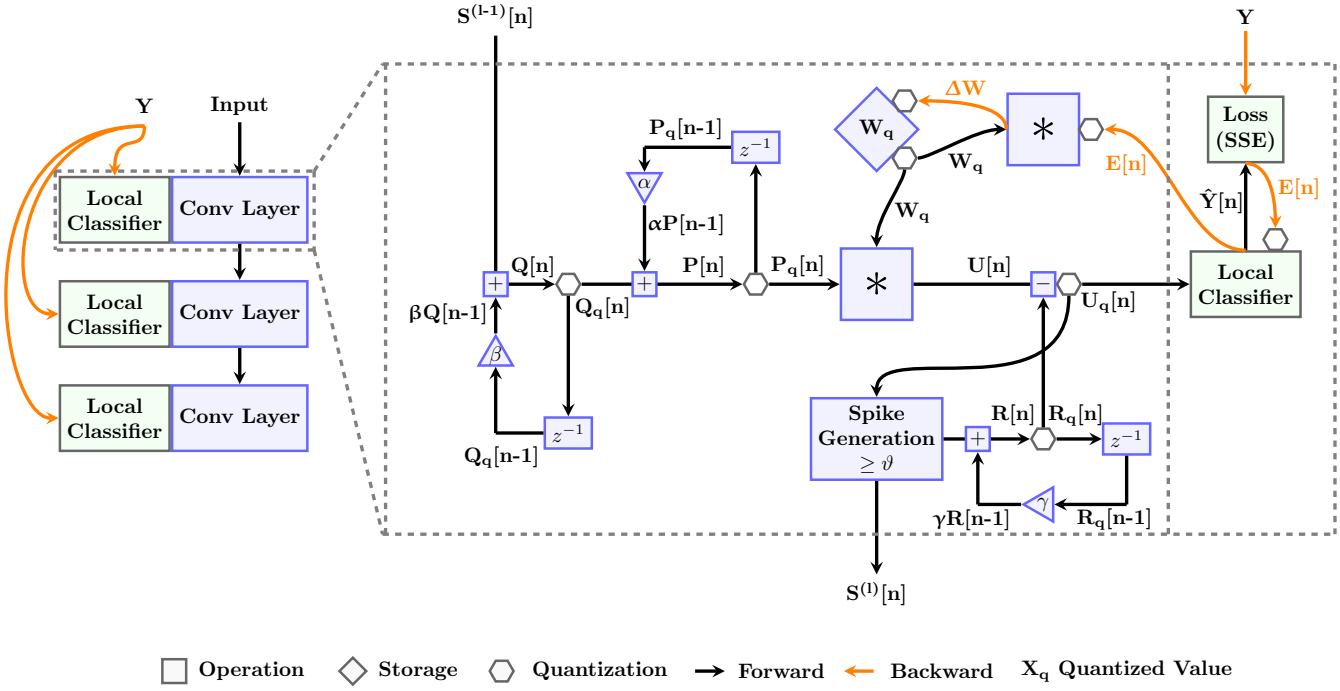

Figure 1: An overview of our implementation of quantization to emulate fixed-point hardware operation. The architecture of the SNN implements local learning at each layer together with internal neural dynamics. Black arrows indicate the flow of data for forward passes and orange arrows indicate the data flow for a backward pass. Hexagons represent stages where we implement software level quantization, we use the subscript q to denote a quantized variable.

randomly by drawing from a uniform distribution. The limits of this distribution are chosen to be the maximum of the heuristic presented in [5], based on the fan-in of a neuron (n_{in}) or the step-size (2^{1-k_S})

$$S \sim U(-L, +L), \quad L = \max\left(1/\sqrt{n_{in}},\ 2^{1-k_S}\right). \tag{5}$$

Since local classifier weights are kept constant, they only need to be quantized at initialization. To account for scaled weights due to decreased precision from fewer bits, the authors in [30] introduce an activation scaling factor ($\rho^{(l)}$) computed for every layer which prevents classifier outputs from exploding. Here, ρ is defined as:

$$\rho^{(l)} = \max\{2^{\text{round}(\log_2(\sigma(k_S)/L))},\ 1\}.$$

Activations of the local classifier are scaled by $\rho^{(l)}$ and then quantized by (4) before they are passed on to be evaluated by the loss function, generating a loss which can be backpropagated. Similar to [30], we scale errors between $[-\sqrt{2}, +\sqrt{2}]$ by dividing errors with $2^{\text{round}(\log_2(\max\{|E|\}}$, which renormalizes the errors while preserving orientation. This scaling is a layerwise operation, discarding all errors smaller than the minimum step size (after scaling), leading to a mild regularization effect. Calculation of the loss function is made computationally efficient by the use of the sum of square error (SSE) loss function.

3.2 Quantizing Neural Dynamics

SNNs implementing DECOLLE, have five primary variables that incur storage and computational costs. First, the spike $S_i^{(l)}$ is a single-bit value that represents the membrane potential U equalling or exceeding a threshold (ϑ). The membrane potential in our model can be directly calculated using P, W, and R, thus not requiring dedicated storage. We explicitly opted against dedicated storage for U since computation are cheaper than storing values and fetching them from memory. Therefore, the main factors involved in determining computational and storage costs are W, P, Q and R as well as the gradients to update W. We address the quantization strategy for these variables in sequence.

The weights are initialized as in equation (5), although the biases are scaled by an factor of 100 following [13]. Since the weights themselves are not subject to any temporal dynamics, we leverage existing algorithms for training NNs with quantized weights [30]. We do not quantize weights differently between the forward and the backward pass since such differences would not result in any actual memory savings for learning on chip, while the energy-savings from reduced precision computation would be minimal when subject to continuous learning. We apply the function $\zeta(x, k)$ in (4) to the weights at every forward pass. Quantizing P, Q and R requires modifications to $\zeta(x, k)$, since the values can be much larger than 1 and by definition will never be negative (not requiring a sign bit). It's possible to analytically determine the maximum

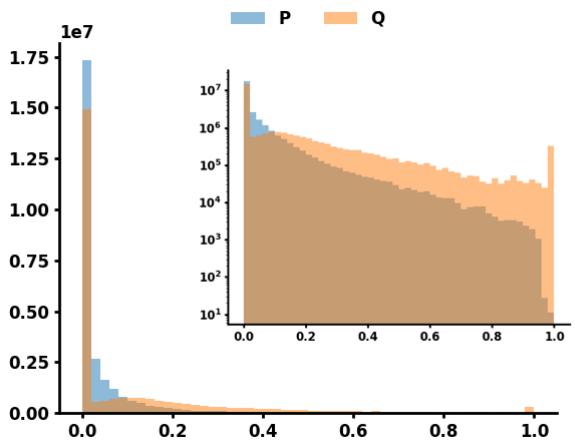

Figure 2: Histogram of P and Q values over 450 training time steps of one batch normalized by their max values excluding zeros. The inner histogram is a log scaled version of the outer histogram. Illustrating the highly skewed distributions of P and Q values, representing challenges for quantization.

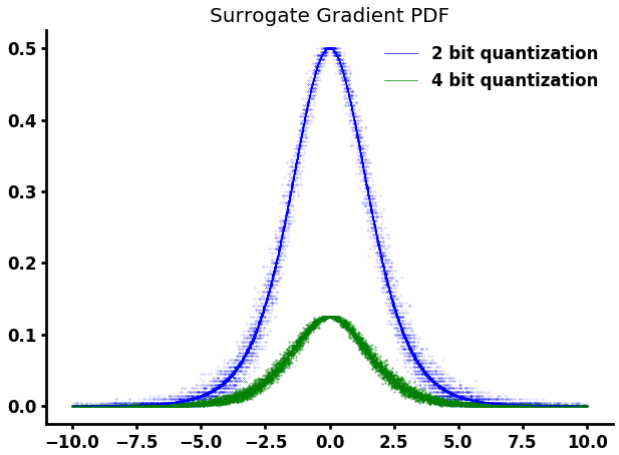

Figure 3: The probability density function (PDF) of two surrogate gradients (using $U[n]$ and the derivative of a sigmoid). Blue line indicates mean for a gradient quantized to 2-bits and green line represents the mean of a gradient quantized to 4-bits. The mean was calculated over 100 samples per value, the dots around the solid lines display the variance around the mean.

value of these variables given the time constant τ. So, we compute the upper bounds for each variable as:

$$\theta_Q = \frac{\tau_{syn}}{1-\beta}, \quad \theta_P = \frac{\tau_{mem} \cdot \theta_Q}{1-\alpha}, \quad \theta_R = \frac{1}{1-\gamma}.$$

Thus, the traces can be quantized within a range of $[0, \theta]$. We can define a new function suited to quantizing synaptic and neural dynamics, denoted by $\zeta^*(x, k, \theta)$, adapting the function $\zeta Q(x, k)$ in equation (4) by removing both clipping and the sign bit

$$\zeta^*(x, k, \theta) = 2^{-k}\text{round}\left[\frac{x}{2^{-k}\theta}\right]\theta.$$

As seen from the histogram in Fig. 2, which shows the distribution of P and Q during training and inference, the precision of of lower P and Q values is important for the performance of the SNN. Figure 2 shows the frequency of normalized values gathered over 450 training steps for both P and Q (in both linear and logarithmic axes for a clear comparison). Most of these values are small, thus the theoretical maximum value is rarely needed. With this in mind, further gains in performance could be derived from reallocating the same bits to cover a lower range of values. Note also that θ_P for commonly used τ's will be larger than θ_Q. In Section 4.1 we will analyze the effect of allocating bits to scaled values of P and Q and the corresponding effect on the performance.

3.3 Quantizing Errors and Gradients

The local classifier provides errors which are quantized to preserve orientation information. Consequently, our gradient (ΔW) quantization focuses primarily on direction. Following [30] we normalize gradients by their maximum absolute value and multiply them by the learning rate η_q. In contrast with learning rates employed in traditional gradient descent, η_q represents a step based learning rate indicating how many steps will be applied to the weights. The gradient quantization implements stochastic rounding [6] to account for small updates on average. The learning rate η_q can thus be larger than 1 to accelerate training and smaller than 1, which would introduce greater stochasticity in the training process but also allow for smaller steps

$$\Delta W = \eta_q g / 2^{\text{round}(\log_2(\max\{|g|\}}$$
$$\Delta W_q = \omega(k_G)\text{sgn}(\Delta W)\left[\lfloor|\Delta W|\rfloor + \text{Bernoulli}(|\Delta W| - \lfloor|\Delta W|\rfloor)\right].$$

Given a $k_{\Delta W}$, we can set the gradient quantization to be more or less stochastic. When $\eta_q = 1$, the weights will be modified by a maximum of $2^{1-k_{\Delta W}}$ if the gradient equals 1 (after normalization), otherwise stochastic updates are initiated. In Fig. 3, the solid line represents the average value of the gradient over 10000 samples, assuming $\eta = 1$ and a surrogate gradient based on the membrane potential. The points represent the sampled variance around the mean. The blue and green color scheme represents the case where $k_{\Delta W}$ is 2 and 4 respectively, showing the difference in update size due to reduced precision. There is minimal stochasticity at the extremes and the midpoint of the distribution, which the intermediate points incur larger variance in the update. This matches our theoretical framework that updates are very likely when they are closely associated to a spike ($U[n]$ close to ϑ) and the least certain when they are only weakly connected to a spike, this also hints at connections to STPD.

3.4 Loss Function for Quantized Training

We define a loss function based on the sum-square-errors functions [30] augmented by the regularizer presented in [13]

Table 1: Settings used for experiments.

Variable	Value	Variable	Value
Δ_t	$1ms$	λ_2	0.0001
τ_{mem}	$5-35ms$	ϑ	0
τ_{syn}	$5-10ms$	$\hat{\beta}_1$	1.5
τ_{ref}	$\frac{1}{0.35}ms$	$\hat{\beta}_2$	0.5
PQ_{cap}	1	η_q	1
U_{scale}	4×10^{-5}	η_{fp}	1.0×10^{-9}
λ_1	0.02	p dropout	0.5
burn-in	$50ms$		

$$L=\sum^{L} SSE(\hat{Y},Y)+\lambda_1\mathbb{E}_i[\sigma_{\mathrm{R}}(U_i^l+0.01)]+\lambda_2\sigma_{\mathrm{R}}(0.1-\mathbb{E}_i[U_i^l]). \quad (6)$$

In equation (6) $\hat{Y}$ stands for the output of a local classifier and $\langle\cdot\rangle$ represents an expectation operation over index i, σ_R is the rectified linear unit (ReLU) activation. Figure 1 illustrates the interplay of all previously described techniques and visualizes a forward (black) as well as backward pass (orange) of SNN training by local classifiers.

4 EXPERIMENTS AND RESULTS

We replicate the SNN simulations in [13], using the architecture composed of 3 convolutional layers with of 7×7 kernels and 64, 128 and 128 filters respectively using the PyTorch framework. Further, we interleaved max pooling layers after layer 1 and 3 and used a dropout layer (with a dropout probability $p=0.5$) at the local classifier to prevent overfitting. To prevent inadvertantly training on the test set, we instead only use the training set through an 80/20 cross-validation split. The validation data was used to judge the SNN performance and pick a configuration (e.g. weights) which resulted in the best trained SNN. This model, is then finally evaluated on the test set.

Table 1 records the parameters used for our experiments. The values for τ_{mem} and τ_{syn} indicate the range over which they were uniformly sampled for every neuron. The U_{scale}, rescales the membrane potential to be mid-range for the sigmoid activation to facilitate learning. The variables $\hat{\beta}_1$ and $\hat{\beta}_2$ are additional scale variables for the weight initialization for the SNN layer weights and local classifier weights respectively. When evaluating the unquantized SNN, we use stochastic gradient descent with a learning rate η_{fp} and a standard mean square error loss function. We evaluated the performance of our SNN and quantization technique on two different data sets: 1) Slow Poker DVS [28] and 2) Gesture DVS [1].

For the poker DVS data set, each recording is an input of size 128×128 belonging to one of four classes. We extracted 968 samples of 500ms from the recordings, equally split over the four classes. The samples were randomly categorized into training (619 sequences), validation (155 sequences), and test set (194 sequences). To process this data set we applied max pooling with a 4×4 kernel to the data, thereby reducing the dimensionality to 32×32. We trained our model for a 100 epochs, dividing the learning rate at epochs 40 and 80 by two.

Table 2: Effect of truncating P and Q, rows with * are in percentages and † indicates that reported values are averages over the last 40 epochs.

PQ Cap*	25	50	75	80	90	100
Test Acc.*	88.89	86.46	86.46	86.81	85.07	87.85
Val. Acc.* ($\mu^\dagger$)	84.14	87.36	83.94	87.24	86.28	84.04
Val. Var. ($\sigma^\dagger$)	0.0476	0.0364	0.0386	0.0451	0.0498	0.0463

Our model training for DVS gesture uses max pooling to downsample the 128×128 down to 32×32. We randomly sample 500 ms sequences for training, and we use 1800ms for testing and validation. We used the same SNN architecture for 222 training epochs with the learning rate halving every 80 epochs.

4.1 Results

First, we show the performance of quantized networks compared to unquantized networks on a given task. Next, we explore the sensitivity of our models towards different quantization levels. Then, we analyze the loss-landscape of quantized networks and compare it to the loss-landscape induced by the unquantized models. Finally, we examine the effect of quantization on the spike activity levels in the SNNs.

Unless otherwise noted, the quantized network has 12-bits allocated to membrane potential (P), 10-bits to synapse state (Q), 2-bits to refractory state (R), 8-bits to weights (W), 10-bits to gradients and local classifier weights. The membrane potential, activations, and errors are all allocated 6-bits (base model). Fig. 5 and 6 show the moving average (length 5) of the validation error of the third layer of the quantized and unquantized SNN on the DVS poker and gesture dataset. We then compute the test error on the model with the lowest reported validation error. In our evaluation on DVS poker (Fig. 5), we are able to achieve a test error of 1.54% for our quantized baseline and no error for its unquantized counterpart. On the DVS gesture dataset, which is a more difficult task, our base model achieved a test accuracy of 12.15% compared to 11.11% from the unquantized model. Note, that due to the shorter training period and our use of cross-validation to pick the best model our results are not directly comparable [13]. Stochastic rounding resulted in up to 2.43% variation in test-error over multiple training runs.

Fig. 2 indicates that smaller values of P and Q occur more frequently. We tested this by training different models with P and Q parameters, truncated to various levels relative to their maximum. Table 2 shows how the model performs over a range of value truncations. Contrary to expectations from traditional ANNs [30], this increased fidelity in representing frequently occurring values does not significantly influence performance.

Given the close performance of quantized SNNs to their unquantized counterparts, we examine the effect of varying the bits allocated to different variables. We vary the number of bits available for the errors and gradients as shown in table 3. Table 3 shows that at this level of quantization, SNNs are relatively insensitive towards minor increases to bits allocated towards gradients and errors. However reductions in the bit-width severely hampers the performance. We also explore the local classifier's tolerance to reduced weight

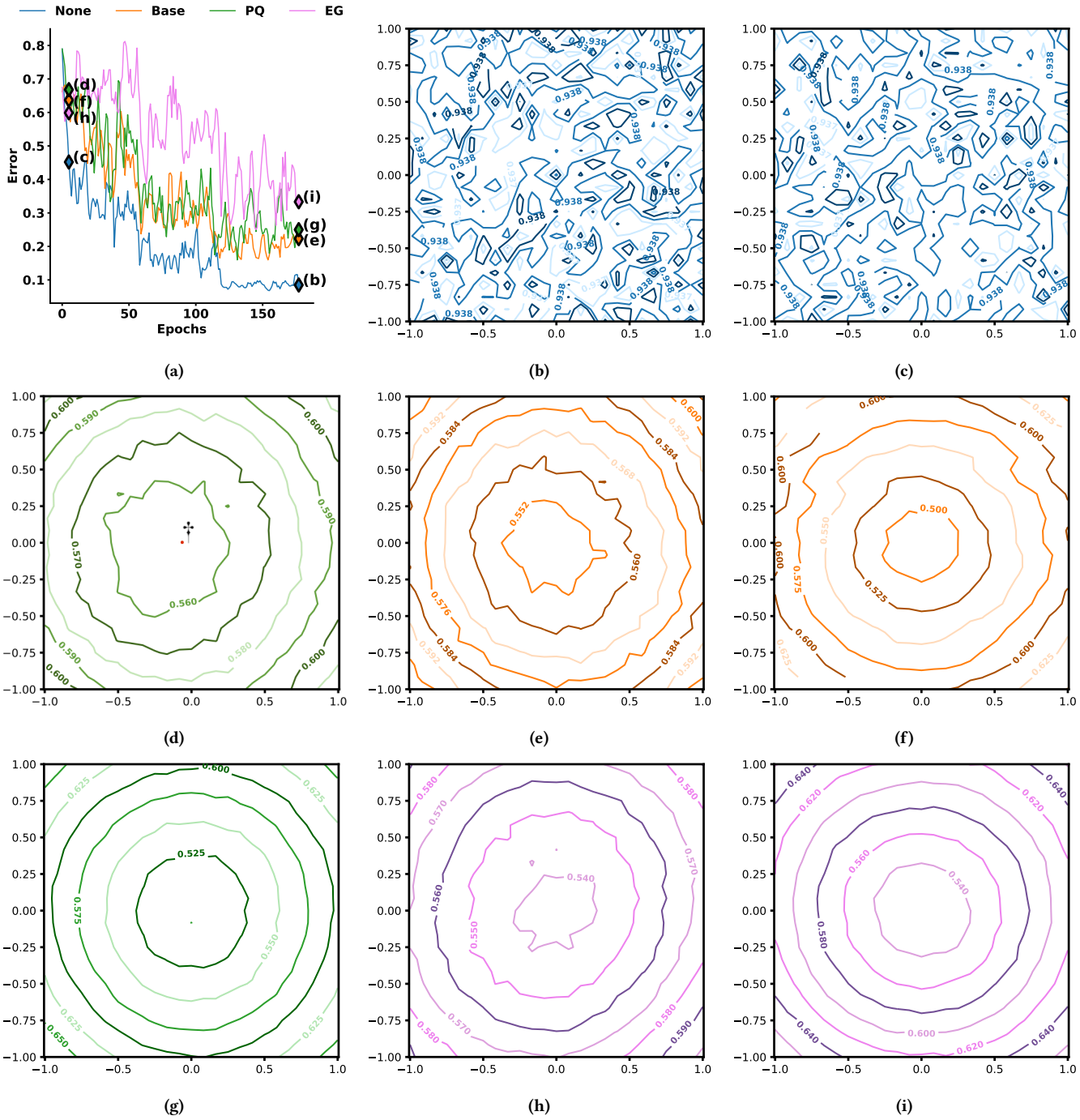

Figure 4: (a) Validation set error over the training process with markers indicating at which points the loss landscape (b)-(i) were extracted. Note *None* (blue) stands for a network with 32-bit single precision floating point, *Base* (orange) for the previously described quantized base model, PQ (green) for base SNN with P and Q further reduced by 2-bits and EG (magenta) for the base model where E and G are set to 8- and 10-bits respectively. † indicates the approximate size of an area in the loss-space in which the model parameters remain invariant when quantized to 2-bit weights.

precision. Since our weights in the forward and backward pass are constrained to have matching signs, setting weights to be 1 bit, results in matching weights for the forward and backward pass.

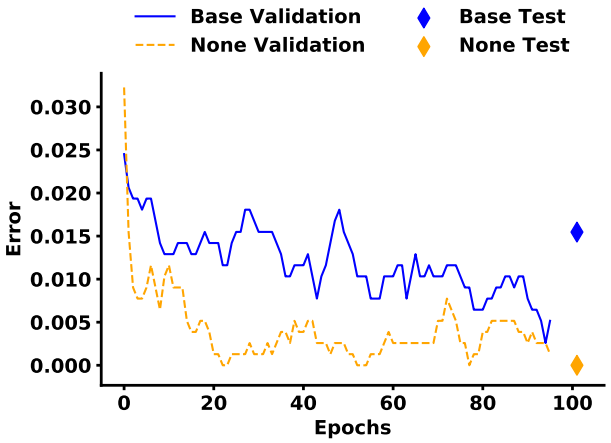

Figure 5: Moving average (length 5) of the validation error on the DVS Poker data set. The blue line represents our base SNN meanwhile the dashed orange line is a non quantized SNN with the same architecture. The diamonds display the test set error.

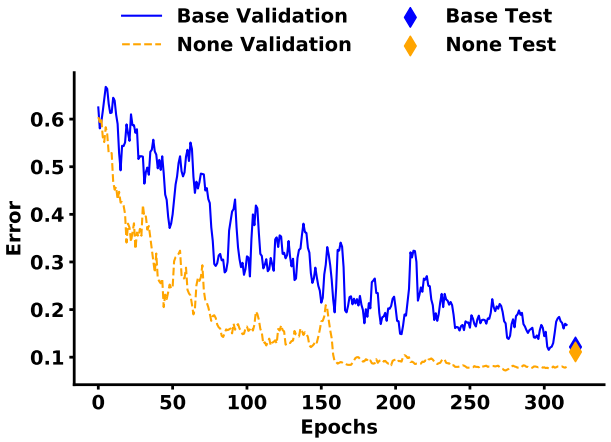

Figure 6: Validation curve (moving average of the validation error) for the DVS gesture data set for a quantized SNN (blue) and a non-quantized (orange) network. The diamond indicated the test set performance of each network respectively.

The number of bits allocated to local classifier weights were observed to have a marginal effect on the accuracy. However, for 2-bit weights, we see a significant decrease in validation set performance despite high accuracy on the training set, suggesting a lack of regularization. However, further investigations are required to better understand the complex trade-offs. Neural dynamics are central to SNN operation, consequently, we investigate the effect of operating neural dynamics at reduced precision. P and Q capture the temporal dynamics as shown in (1), reducing their precision has a gradual

Table 3: Effect of widths bit for various variables on accuracy, all accuracies bases in the test set and given in percent.

Error and Gradient Bits							
Bits	6/8	7/9	8/10	9/11	**10/12**	11/13	12/14
Accuracy	40.62	46.18	59.38	74.65	**87.85**	87.15	87.85
Local Classifier Weight Bits							
Bits	2	3	4	5	**6**	7	8
Accuracy	76.04	86.11	88.54	82.29	**87.85**	84.03	84.72
P and Q Bits							
Bits	8/6	9/7	10/8	11/9	**12/10**	13/11	14/12
Accuracy	77.43	74.31	78.47	85.76	**87.85**	84.72	82.99

effect on the test error. Reducing the bit-precision below 11/9-bits reduces the SNN accuracy as seen in Table 3.

For insight into the effect of quantization on SNN training we visualize the training loss landscape with the methodology devised in [15]. Figure 4 illustrates several training loss landscapes and Figure 4a marks the epoch at which the models are evaluated from. The loss landscapes for floating point SNNs in Figure 4b and 4c are rather chaotic, which can in part be due to the small learning rate and the floating-point granularity of all parameters which allows for greater differentiation and a more fine-grained loss landscape. Looking at the loss landscape of quantized networks (Figure 4e, 4f, 4d, 4g, 4h and 4i) we can clearly see, fairly sized optima in the middle, which in theory should be easy to solve for. This matches the training curves of our models which suggest fast training, at least regarding training loss and accuracy. The training loss and accuracy however are not always indicative of the test performance. Especially at smaller learning rates in the later epochs, which benefit the test set performance greatly. The contrast between early and later stage loss landscapes might reveal why. Early stage loss landscapes (Figure 4e, 4d and 4h) do not appear as smooth as later stage landscapes, which appear to be better conditioned. This hints that minimas at later stages, which are less stochastic and better for generalization [25]. The red area annotated with the † in Figure 4d shows the approximate size of an area occupied by a model quantized with 2-bit weights, projected onto the loss landscape.

Finally, we examine the effect of quantization on SNN spiking activity. Our central aim with quantization is to reduce the energy cost of storing and computing the SNN model. However, reduced precision in computing can also trade-off with increased spiking communication. Figure 7 shows the trade off between quantization, accuracy, and quantization induced increased spike activity. The figure reflects the effect, on SNN activity, of provisioning the errors and gradients and the neural and synpatic traces (P and Q) with varying bit-widths. We use darker colors to denote fewer bits and lighter colors to denote more bits allocated to those variables. We can see that increasing the quantization of parameters (fewer bits) leads to an increase in SNN spiking activity.

5 CONCLUSION

In summary, we analyzed the effects of quantizing neural dynamics on a hardware-friendly SNN model. We explore the performance of

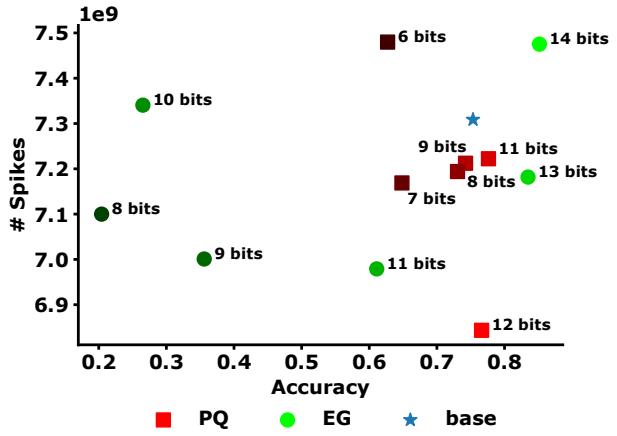

Figure 7: Quantization induced spike activity accuracy tradeoff. Darker symbols indicate fewer bits available to the respective variable, meanwhile lighter symbols represent settings with broader bit widths.

our quantized SNNs on two data set (DVS Poker and DVS Gesture) and analyzed the dependency of the test error on the number of bits allocated towards different factors that control the neural and learning dynamics: temporal dynamics, gradients, errors, and weights for the local classifier. Until a transition point is encountered in performance (highlighted in Tab. 3) computing neural dynamics at reduced precision had the most impact on SNN performance, meanwhile the precision with which error, gradient, and local classifier calculations are performed had mostly marginal effects. Beyond the transition point, SNN performance deteriorates significantly. Given our base model, we showed that through quantization we can roughly reduce the memory by 73.78% while only incurring a test error of 1.54% for the DVS Poker and incurring a 1.04% deterioration in test accuracy on the DVS Gesture data set.

REFERENCES

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. 2017. A Low Power, Fully Event-Based Gesture Recognition System. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[2] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 1 (2018), 82–99.

[3] Tobi Delbruck and Manuel Lang. 2013. Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Frontiers in neuroscience* 7 (2013), 223.

[4] Wolfram Gestner and W Kistler. 2002. Spiking neuron models. *Cambridge University* (2002).

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[6] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[7] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. 2011. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience* 5 (2011), 73.

[8] G. Indiveri and Y. Sandamirskaya. 2019. The Importance of Space and Time for Signal Processing in Neuromorphic Agents: The Challenge of Developing Low-Power, Autonomous Agents That Interact With the Environment. *IEEE Signal Processing Magazine* 36, 6 (2019), 16–28.

[9] Eugene M Izhikevich. 2003. Simple model of spiking neurons. *IEEE Transactions on neural networks* 14, 6 (2003), 1569–1572.

[10] Xin Jin, Alexander Rast, Francesco Galluppi, Sergio Davies, and Steve Furber. 2010. Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[11] Yingyezhe Jin, Wenrui Zhang, and Peng Li. 2018. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*. 7005–7015.

[12] Jacques Kaiser, Alexander Friedrich, Juan Camilo Vasquez Tieck, Daniel Reichard, Arne Rönnau, Emre Neftci, and Rüdiger Dillmann. 2019. Embodied Event-Driven Random Backpropagation. *CoRR* abs/1904.04805 (2019). arXiv:1904.04805 http://arxiv.org/abs/1904.04805

[13] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. 2018. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *arXiv preprint arXiv:1811.10766* (2018).

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[15] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the Loss Landscape of Neural Nets. In *Neural Information Processing Systems*.

[16] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications* 7, 1 (2016), 1–10.

[17] Wolfgang Maass. 2015. To spike or not to spike: that is the question. *Proc. IEEE* 103, 12 (2015), 2219–2224.

[18] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[19] J Parker Mitchell, Grant Bruer, Mark E Dean, James S Plank, Garrett S Rose, and Catherine D Schuman. 2017. NeoN: Neuromorphic control for autonomous robotic navigation. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE, 136–142.

[20] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. 2018. Deep supervised learning using local errors. *Frontiers in neuroscience* 12 (2018), 608.

[21] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. 2019. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948* (2019).

[22] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. 2019. Surrogate Gradient Learning in Spiking Neural Networks. *Signal Processing Magazine, IEEE* (2019).

[23] Emre O Neftci, Bruno U Pedroni, Siddharth Joshi, Maruan Al-Shedivat, and Gert Cauwenberghs. 2016. Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in neuroscience* 10 (2016), 241.

[24] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. 2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience* 11 (2017), 682.

[25] Sungho Shin, Yoonho Boo, and Wonyong Sung. 2020. SQWA: Stochastic Quantized Weight Averaging for Improving the Generalization Capability of Low-Precision Deep Neural Networks. arXiv:2002.00343 [cs.LG]

[26] Sumit Bam Shrestha and Garrick Orchard. 2018. SLAYER: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*. 1412–1421.

[27] Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik. 2019. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *arXiv preprint arXiv:1912.01268* (2019).

[28] Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. 2017. *"SLOW-POKER-DVS Database"*. "http://www2.imse-cnm.csic.es/caviar/SLOWPOKERDVS.html"

[29] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

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

[31] Amirreza Yousefzadeh, Timothée Masquelier, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. 2017. Hardware implementation of convolutional STDP for on-line visual feature learning. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.

[32] Amirreza Yousefzadeh, Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. 2018. On practical issues for stochastic stdp hardware with 1-bit synaptic weights. *Frontiers in neuroscience* 12 (2018).

[33] Friedemann Zenke and Surya Ganguli. 2018. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation* 30, 6 (2018), 1514–1541.