Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing

Peter U. Diehl[1], Daniel Neil[1], Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer
Institute of Neuroinformatics, University of Zurich and ETH Zurich
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland

***Abstract*—Deep neural networks such as Convolutional Networks (ConvNets) and Deep Belief Networks (DBNs) represent the state-of-the-art for many machine learning and computer vision classification problems. To overcome the large computational cost of deep networks, spiking deep networks have recently been proposed, given the specialized hardware now available for spiking neural networks (SNNs). However, this has come at the cost of performance losses due to the conversion from analog neural networks (ANNs) without a notion of time, to sparsely firing, event-driven SNNs. Here we analyze the effects of converting deep ANNs into SNNs with respect to the choice of parameters for spiking neurons such as firing rates and thresholds. We present a set of optimization techniques to minimize performance loss in the conversion process for ConvNets and fully connected deep networks. These techniques yield networks that outperform all previous SNNs on the MNIST database to date, and many networks here are close to maximum performance after only 20 ms of simulated time. The techniques include using rectified linear units (ReLUs) with zero bias during training, and using a new weight normalization method to help regulate firing rates. Our method for converting an ANN into an SNN enables low-latency classification with high accuracies already after the first output spike, and compared with previous SNN approaches it yields improved performance without increased training time. The presented analysis and optimization techniques boost the value of spiking deep networks as an attractive framework for neuromorphic computing platforms aimed at fast and efficient pattern recognition.**

I. Introduction

DEEP neural network architectures, such as convolutional neural networks (ConvNets) [1] and fully-connected feed-forward neural networks [2], are currently the most successful architectures for natural image classification. They have achieved record-breaking results for problems such as handwriting recognition [2], scene labeling [3], the CIFAR benchmark [4], the ImageNet benchmark [5], and many others. Deep neural network architectures, which are loosely inspired by hierarchies of cortical visual information processing [6], have seen increasing success in recent years due to the availability of more powerful computing hardware, larger datasets, and improved training algorithms, which has enabled the training of much deeper networks, while avoiding problems of overfitting [7]. Despite their successes, the substantial computational cost of training and running deep networks has created a need for specialized hardware acceleration and new computational paradigms to enable the use of deep networks for real-time practical applications. Spiking neural networks (SNNs) are a primary candidate for enabling such acceleration, and in this paper we introduce a new optimization method for spiking deep architectures - both fully-connected feed-forward networks and ConvNets - that can achieve higher performance levels than previous spiking solutions, while achieving lower latencies and requiring fewer operations than methods based on conventional computing.

In recent years, spiking deep neural networks have become an increasingly active field of research. This has been driven both by the interest to build more biologically realistic neural network models, and by recent improvements and the availability of larger-scale neuromorphic computing platforms, which are optimized for emulating brain-like spike-based computation in dedicated analog or digital hardware [8], [9], [10], [11]. Neuromorphic platforms can be orders of magnitude more efficient in terms of power consumption compared to conventional CPUs or GPUs for running spiking networks, and often permit distributed and asynchronous event-based computation, thereby improving scalability and reducing latencies. Furthermore, event-driven neuromorphic systems focus their computational effort on currently active parts of the network, effectively saving power on the rest of the network. They are therefore attractive as platforms to run large-scale deep neural networks in real-time. These platforms are ideally driven by input from neuromorphic sensors such as silicon retinas [12] or cochleas [13], which create sparse, frame-free, and precisely timed streams of events, with substantially reduced latencies compared to frame-based approaches. Earlier work on spiking deep networks has thus focused on fast classification in event-based vision systems with ConvNets and Deep-Belief Networks (DBNs) [14], [15], [16].

Training of spiking deep networks typically does not use spike-based learning rules, but instead starts from a conventional ANN, fully trained with backpropagation, followed by a conversion of the rate-based model into a model consisting of simple spiking neurons. Theory has shown that SNNs are at least as computationally powerful as their analog counterparts [17], but practically it has proven difficult to come up with equivalent solutions. One approach used for example by [16] is to train spiking DBNs by using the Siegert mean-firing-rate approximation of leaky integrate-and-fire (LIF) neurons to approximate probabilities during training. Another approach, used in [18], requires tuning of parameters such as leak and refractory period in the spiking network. In both cases, the spiking network suffers from considerable losses in classification accuracy, when compared to a non-spiking network of similar architecture.

Recently, [19] proposed a method for spiking ConvNet

[1]The authors contributed equally to this work.
Email: {diehlp,daneil}@student.ethz.ch

conversion that achieves significantly better performance than previous approaches, by taking the characteristic differences of spiking and non-spiking networks into account. The main challenges are the representation of negative values and biases in spiking neurons, which are avoided by using rectified linear units (ReLUs) during training, and setting all biases to zero. Furthermore, the typical max-pooling operations of ConvNets are replaced by spatial linear subsampling. Still, the resulting spiking ConvNet after conversion suffers from a small loss of performance. In this work we present a more detailed analysis of the sources of such performance losses in spiking networks, and present several tools for optimization. We find that if SNNs are driven in the right regime, near-lossless conversion is possible, and additionally very fast classification is possible based on only a few output spikes.

II. Neural Network Architectures

A. ReLU-Based Feed-Forward Neural Networks

In fully connected feed-forward Neural Networks (FCNs), all neurons in the preceding layer are fully connected to the subsequent layers, with no intra-layer connections. Recent competitive results (e.g. [2]) have renewed the interest in this architecture. Initializing the extremely high-dimensional weight vectors of FCNs in a good regime that preserves the error gradient, and regularizing the network to prevent overfitting, allows very high performance on standard test sets. The most recent improvements have come with the introduction of the dropout training technique (see section II-C) in combination with rectified linear units (ReLUs) [20]. ReLUs are a type of nonlinearity which is applied to the weighted sum of inputs, and is described by

$$x_i = \max\left(0, \sum_j w_{ij} x_j\right) \quad , \tag{1}$$

where x_i is the activation of unit i, w_{ij} is the weight connecting unit j in the preceding layer to unit i in the current layer, and x_j is the activation of unit j in the preceding layer. By successively updating all the activations of a current layer based on the activations of the previous layer, the input is propagated through the network to activate the output label neurons.

Training proceeds according to standard error backpropagation, successively propagating an error gradient backwards through the layers by computing local derivatives to update individual weights and minimize the error. In these networks, the training process adjusts the randomly-initialized weight matrix describing the connections between the layers to minimize the overall error through stochastic gradient descent. Further details can be found in [20].

B. Convolutional Neural Networks

ConvNets [1] are multi-layered feed-forward architectures in which feature detectors take the form of simple convolution kernels. Typically, a convolutional neural network is composed of alternating layers of convolution and spatial subsampling, with nonlinearities between subsequent iterations. Here, a convolutional layer generates a number of feature maps, which are obtained by convolving patches of the preceding layer with a set of kernels $\{W^k, k = 1 \dots n\}$. The resulting maps $\{x^k, k = 1 \dots n\}$ are given by

$$x^k = f\left(\sum_l W^k * x^l + b^k\right) \quad , \tag{2}$$

where f is the neuron's nonlinear activation function, x^l is the activation of the units of the preceding layer's activation map l, the $*$ symbol denotes a 2D valid-region convolution, and b^k is a bias term. As above, we use ReLU, as described in equation 1 as the activation function f. The kernels only respond to a small rectangular patch of inputs, specified by W^k, which are repeated and moved over the whole input map. Convolutional layers are often followed by subsampling or pooling layers, whose units combine the responses of multiple feature detectors into one. While many choices exist for pooling layers, an averaging kernel is used here to enhance the portability of this ConvNet to an SNN. The activation of this averaging layer is identical to equation 2, except that the kernel weights W^k_{ij} are fixed to $1/\mathrm{size}(W^k)$, where $\mathrm{size}(W^k)$ is the number of pixels in the kernel.

ConvNets reduce the data dimensionality by alternating between convolution and subsampling layers, while producing increasingly abstract features to describe the input. Additionally, the lower number of weights in the ConvNet compared to fully connected archictectures reduces the problem of overfitting. The output of the ConvNet is the concatenation of all feature maps in the final layer, which forms the input to a simple fully-connected neural network trained as a classifier. Just like FCNs, training uses stochastic gradient descent via backprop to adjust the weights in the network, using weight sharing to learn W^k in the convolution layers, together with the weights for the final output layer. Further details can be found in [1].

C. Dropout

Overfitting is a well-known problem of large and deep neural networks. A successful method to avoid this problem is to employ regularizers, such as the recently proposed dropout technique [20]. Dropout randomly disables input units during learning, and thus avoids overspecialization and co-adaptation of hidden units. In this work, dropout is used in the activation function as a mask that randomly disables ReLU activations on a given trial, thereby effectively increasing the overall robustness of the network. A ReLU activation function with dropout is given by

$$x_i = \begin{cases} \max\left(0, \sum_j w_{ij} x_j\right) & \text{with probability } d_r \\ 0 & \text{otherwise}\,, \end{cases} \tag{3}$$

where the variables are as in equation 1 with the addition of a dropout rate d_r. At every training iteration, a new random decision is made for each unit. In practice, a d_r value of 0.5 is often used to turn off half of the connections randomly in each training step.

III. Spiking Neural Networks

A. Background

In a conventional ANN, a whole input vector is presented at one time, and processed layer-by-layer, producing one output value. In an SNN, however, inputs are typically presented as streams of events, and neurons integrate evidence during

the presentation, creating spikes to communicate information to subsequent layers, ultimately driving firing of output neurons which sum evidence over time. There are significant advantages of this approach: pseudo-simultaneity of input and output can be achieved [21], time-varying inputs can be more efficiently processed [16], and more efficient computation on specialized hardware can be accomplished [22].

The spiking neuron model used for this work is the simple integrate-and-fire (IF) model. The evolution of the membrane voltage v_{mem} is given by

$$\frac{dv_{\mathrm{mem}}(t)}{dt} = \sum_i \sum_{s \in S_i} w_i \, \delta\left(t - s\right) \quad , \tag{4}$$

where w_i is the weight of the ith incoming synapse, $\delta(\cdot)$ is the delta function, and $S_i = \{t_i^0, t_i^1, \ldots\}$ contains the spike-times of the ith presynaptic neuron. If the membrane voltage crosses the spiking threshold v_{thr}, a spike is generated and the membrane voltage is reset to a reset potential v_{res}. In our simulations, this continuous-time description of the IF model is discretized into 1 ms timesteps.

B. Spiking Network Conversion

In previous work, a significant performance gap was reported between the state-of-the-art achieved by conventional ANNs and spiking implementations [16], [19]. Here we lay out a framework to facilitate the conversion of deep ANNs to SNNs, and to reduce the performance loss during this conversion. The conversion method used here is an extension of the one suggested in [19], extended to include our novel normalization methods and the analysis of firing rates and thresholds.

We start with observations about the relationships of ANNs using ReLUs and spiking networks. Firstly, the ReLU can be considered a firing rate approximation of an IF neuron with no refractory period [19], whereby the output of the ReLU is proportional to the number of spikes produced by an IF neuron within a given time window. ReLUs are also advantageous during training as their piecewise constant derivative leads to weight updates of a particularly simple form. Secondly, for classification tasks, only the maximum activation of all units in the output layer is of importance, allowing the overall rate to be scaled by a constant factor. Finally, without a bias to provide an external reference value, the relative scale of the neuron weights to each other and to the threshold of the neuron are the only parameters that matter. This gives rise to the following recipe for converting deep ANNs to SNNs:

1) Use ReLUs for all units of the network.
2) Fix the bias to zero throughout training, and train with backpropagation.
3) Directly map the weights from the ReLU network to a network of IF units.
4) Use weight normalization (see section III-C) to obtain near-lossless accuracy and faster convergence.

These suggestions work for both the case of fully-connected networks and ConvNets. Once the ReLUs in the artificial neural network after training have been replaced by IF neurons, a loss of performance for a fixed simulation duration can come from three factors:

1) The unit did not receive sufficient input to cross its threshold, meaning its rate is lower than it should be;
2) The unit received so much input that the ReLU model predicts more than one output spike per timestep. This can happen either because there are too many input spikes in one timestep or if some of the input weights are higher than the neuron threshold.
3) Due to the probabilistic nature of the spiking input, it can happen that a set of spikes over- or under-activate a specific feature set due to non-uniformity of the spike trains.

Reducing the simulation timestep can help to reduce the number of input spikes per timestep, and increasing the simulation duration will help to avoid insufficient activation. However, all factors can be addressed by finding the right balance of spiking thresholds, input weights and input firing rates. Specifically, high spiking thresholds (or low input weights) decrease the error due to the over-activation and non-ideal spike trains, while increasing errors due to the under-activation factor and vice-versa. Note that only the ratio of spiking threshold to input weights determines the amount of integrated evidence until a spike is fired but not their individual values. Instead of hand-tuning the parameters, here we present a more rigorous approach towards adjusting the network weights (and thereby the ratio of spiking threshold to input weights), i.e. to calculate rescaling factors for the weights which reduce the errors due to the three causes described above.

C. Weight Normalization

A key contribution of this work is a novel weight normalization procedure that puts the network into a regime where the above problematic factors are avoided.

We present two possible ways to normalize the network weights, which ensure that activations are sufficiently small to prevent the ReLU from overestimating output activations. The safest, most conservative method is to consider all possible positive activations that could occur as input to a layer, and rescale all the weights by that maximum possible positive input. If the maximum positive input can only cause one spike, then the network will never need to produce more than one spike at once from the same neuron. By doing so, the resulting spiking networks become robust to arbitrarily high input rates and completely eliminate losses due to too many inputs. Unfortunately, this means that evidence integration in order to generate a spike might require much more time. If a high classification performance is required and longer sampling times are acceptable, this is the preferred method of finding the right weight scaling. This approach, outlined in algorithm 1, is referred to as *model-based normalization* because it requires only knowledge of the network weights.

Alternatively, the training set can be used to estimate typical activations within the network, rather than assuming the worst-case scenario of maximum positive activation. In our experiments, it was observed that this scaling factor is much less conservative. It preserves nearly all the accuracy but requires dramatically less evidence integration time. For this approach, after training a ReLU network, the training set is propagated through the neural network and the ReLU activations are stored. Then, the weights are normalized according

Algorithm 1: Model-Based Normalization

```
for layer in layers:
    max_pos_input = 0
    # Find maximum input for this layer
    for neuron in layer.neurons:
        input_sum = 0
        for input_wt in neuron.input_wts:
            input_sum += max(0, input_wt)
        max_pos_input = max(max_pos_input, input_sum)
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / max_pos_input
```

Algorithm 2: Data-Based Normalization

```
previous_factor = 1
for layer in layers:
    max_wt = 0
    max_act = 0
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            max_wt = max(max_wt, input_wt)
        max_act = max(max_act, neuron.output_act)
    scale_factor = max(max_wt, max_act)
    applied_factor = scale_factor / previous_factor
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / applied_factor
    previous_factor = scale_factor
```

to the maximum possible activation within the training set, so that this case would emit only a single spike. Additionally, this normalization requires taking into account the maximum single input weight as well, since otherwise a single spike could carry so much weight that the receiving neuron would need to spike multiple times within one timestep. While this is not a strong guarantee that performance can be maintained on the test set, the training set should be representative of the test set and results show this approach to be highly effective. This normalization method is especially suitable if both short latencies and high accuracy are required since a practically good tradeoff between those conflicting goals is found. Pseudocode for this approach is shown in algorithm 2 and is referred to as *data-based normalization*, since the weights are scaled according to actual activations of the network in response to data.

IV. Experimental Setup

A. Dataset

Due to its ubiquity in machine learning, the MNIST dataset of handwritten digits was chosen for this investigation. The training set consists of 60,000 individual handwritten digits collected from postal codes, each labeled 0-9 for the individual 28x28 pixel grayscale images. The test set consists of 10,000 digits. The highest reported accuracy on this task using a single network and without extending the data set is 99.55% and was achieved using maxout networks [23]; the highest reported accuracy of a spiking implementation prior to this work is 98.30% [24] which was achieved using spiking ConvNets.

B. Architectures

The code used to train and convert the networks in this paper is a modification of the Matlab DeepLearnToolbox [25] and can be found online.[2] Two main architectures were used in this work. First, to prove the efficacy of these networks for a straightforward typical neural network, a four layer fully-connected neural network was trained. Described in terms of the number of neurons in the network, this 784-1200-1200-10 network has two hidden layers of size 1200 units, with all neurons fully connected between the layers. Five networks were trained using a fixed learning rate of 1, momentum of 0.5, a batchsize of 100, 50 epochs of training, 50% dropout, and weights randomly initialized uniformly between -0.1 and 0.1. After training, the best-performing fully-connected neural network achieves a classification accuracy of 99.87% on the MNIST training set and 98.68% on the MNIST test set.

The second architecture is a 28x28-12c5-2s-64c5-2s-10o ConvNet. The input image is 28x28, followed by 12 convolutional kernels of size 5x5, followed by a 2x2 averaging subsampling window. This convolution process is repeated in a second stage with 64 maps of size 5x5, followed by a 2x2 averaging of the network. These final features are vectorized and fully connected to a 10-node output layer, where each of the 10 nodes represents one of the ten digit classes. The training process used a fixed learning rate of 1, a batchsize of 50, no momentum, 50% dropout of the kernels, zero bias, and 50 epochs of training. We did not use any distortions to extend the original data set. The resulting ConvNet achieves 99.19% training accuracy and 99.14% test accuracy.

The best-performing ReLU network from each of the training methods described above was then selected and the weights were transferred directly to a spiking IF network. A grid search of input rates (25, 50, 100, 200, 400, 1000 Hz) and thresholds (0.25, 0.5, 1, 2, 4, 10, 20) was then performed to determine the spiking networks with the best classification performance. This performance was compared to that of the data- and model-normalized networks of default threshold.

For the FCN, the model-based normalization scaled down the weights in each layer significantly: each layer's weights were multiplied according to algorithm 1, downscaling the layer weights by factors of 0.08 and 0.045. Model-based weight normalization was not applied to the output layer for either FCNs or ConvNets. Unlike the aggressive decrease of the weights due to model-based normalization, the data-based normalization (algorithm 2) scaled the weights by factors of 0.37, 1.25, and 0.8, only adjusting the network slightly but making it more robust to high input rates. In the ConvNet the scaling factors of the weights of the convolutional layers were 0.1657 and 0.1238 for the model-based normalization. Using the data-based normalization, the scaling factors 0.1657, 1.0021 and 1.19 were applied to the convolutional layers and the output layer, respectively. Note that the output layer weights are increased due to too little activation for the training set.

C. Spiking Input

The intensity values of the MNIST images were normalized to values between 0 and 1. Based on those intensity values,

[2] http://github.com/dannyneil/spiking_relu_conversion

TABLE I: Comparison of classification accuracy for different network architectures and conversion mechanisms.

Network Type	Input Rate	Thr.	Accuracy
ConvNets			
ReLU Rate-Based	–	–	99.14%
IF Network	1000 Hz	20.0	99.12%
Data-Norm. Net	400 Hz	1.0	99.10%
Model-Norm. Net	1000 Hz	1.0	99.11%
FC Networks			
ReLU Rate-Based	–	–	98.68%
IF Network	200 Hz	4.0	98.48%
Data-Norm. Net	1000 Hz	1.0	98.64%
Model-Norm. Net	1000 Hz	1.0	98.61%

Poisson distributed spike trains were generated for each image pixel with firing rates proportional to the pixel's intensity value. For further details, see [16].

V. Results

The overall effectiveness of converting custom ReLU networks to SNNs is summarized in Table I. The first network in each section is the original trained ReLU network; its performance is the target performance of the spiking networks. Next, is the best-performing spiking IF network after a grid search of parameters, followed by the data-normalized network with the default threshold; and the model-normalized network. Note that the data-normalized network shows nearly the same performance as the original ReLU network without choosing hyperparameters or sweeping parameters for ideal performance. To obtain these results we simulated the spike-based networks for 0.5 s for each input image. However, in practice, comparable performance can be achieved already after tens of milliseconds of simulated time, as discussed in section V-C.

A. Conversion and Parameter Choices

The activations presented in Fig. 1 describe example responses from the different layers for different parameters of the FCN. The top row shows responses from the layers of the ReLU-based network in response to the input. Note that each image in this plot is individually scaled, but the relative activations of the neurons within the map should ideally match across all rows. While the overall activation structure is well preserved, small differences occur mostly when neurons fire at lower rates.

B. Accuracy

Figure 2 shows the classification error and the number of spikes in the network (without input spikes) of the spiking ConvNet (upper plots) and the spiking fully-connected Network (lower plots) for a range of input firing rates and firing thresholds of the IF neurons. All performances are averaged over five simulations and all spike numbers are averaged over two simulations, using the same network but different input Poisson spike trains for each run. The highest performance of

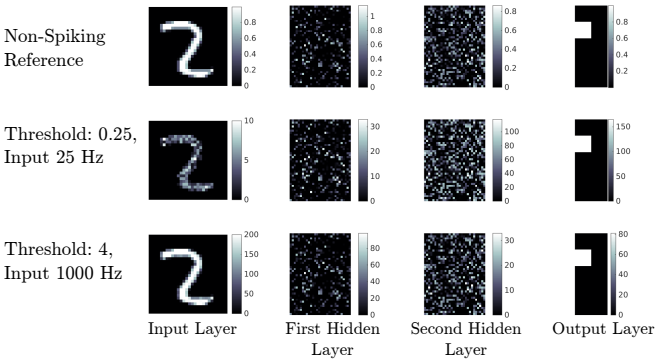

Fig. 1: Comparison of activations between ReLU-based fully-connected network and non-normalized spiking network variants with different thresholds and input rates. The figure shows the accumulated spike count over 200 ms of simulation time. Ideally, the images in the bottom two rows should resemble a scaled version of the top row. Images shown here are scaled individually due to the unbounded upper range of the ReLU.

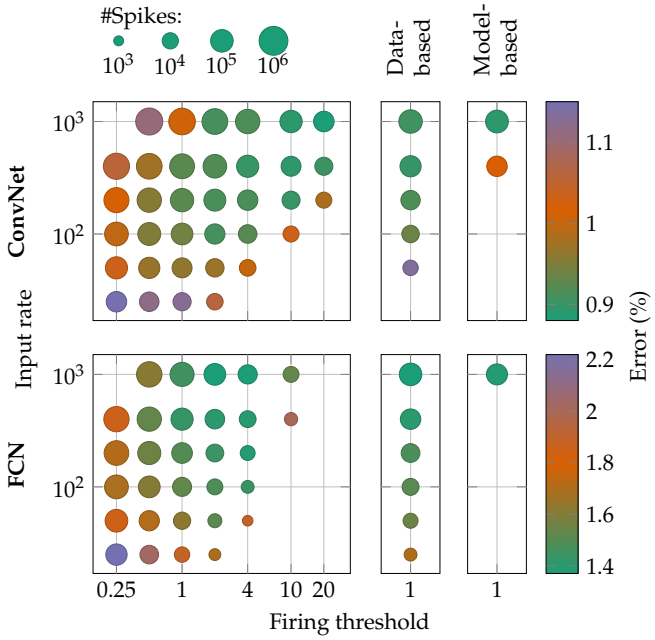

Fig. 2: Classification performance and number of spikes produced for different architectures as a function of the input rate and the firing threshold. Upper panels show results for ConvNets, lower panels for FCN. The color of each circle represents the mean accuracy on the MNIST test set (averaged over 5 trials), using an integration time of 0.5s (500 timesteps) for every input example. The size of the circle corresponds to the average number of spikes generated by the whole network per example presentation. The panels on the right show the same data for the normalized networks, whereby the threshold was fixed at 1 for all experiments. Parameter sets that led to test errors greater than 1.15 (ConvNet) or 2.2 (FCN), respectively, are not displayed.

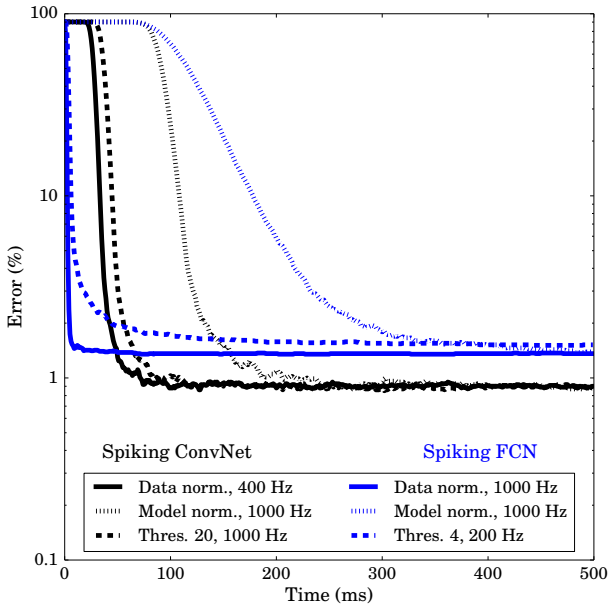

Fig. 3: Classification error over time. Black curves show results for ConvNets and blue curves show the results for fully-connected networks. Solid lines denote the error of data-normalized networks, dashed lines denote the error of model-normalized networks. Dotted lines denote the error for the best parameter set found from the 2D grid (figure 2). All networks except the model-normalized ones show very low error before 100 ms. Fully-connected data-normalized networks are close to their peak accuracy after only 6 ms (1.74% error).

the spiking ConvNet was 99.12% for a first layer IF threshold of 20 and an input rate of 1000 Hz (upper left plot). The best performance of the spiking fully-connected network was 98.48% which was achieved using a threshold of 4 and a 200 Hz input rate, see lower left 2D plot. Generally, there is a trade-off between increasing the threshold to integrate more spikes before propagating the detection of a feature, and decreasing the threshold to reduce the sampling time necessary to produce a sufficient number of spikes to minimize the error due to the discretization of the transmitted messages. For low thresholds, higher performances are achieved using intermediate input rates, whereas for high thresholds, high input rates give better results. The number of spikes generated within the network (including spikes generated at the output layer) increases for higher input spiking rates and decreases for higher spiking thresholds. Surprisingly, the number of spikes generated within the fully-connected network and the ConvNet are comparable, although the fully-connected network uses about 60% more synapses than the ConvNet.

The results for the data-normalized and the model-normalized forms of both the spiking fully-connected network and the spiking ConvNet are shown on the four rightmost panels of Fig. 2. For both network types, the data-normalized networks show very high accuracies over a broad range of input firing rates. In contrast, model-normalized networks show a good performance only for high input rates for the spiking ConvNet. The reason for this can be seen in Fig. 3, which shows the classification error as a function of evidence integration time for the data-normalized networks, the model-normalized networks and the spiking networks with the best parameter set of the 2D plot. Due to the increased thresholds in each layer of the model-normalized network, the sampling time has to be increased to converge to a solution. On the other hand, both data-normalized networks converge faster than the best corresponding networks found in the grid search. This shows that data-based weight normalization is an effective method to obtain fast and accurate spiking deep networks.

C. Convergence Time

One of the reasons to use SNNs is their configurability. If high accuracy is desired, high spiking thresholds help to improve the accuracy; if short latencies are important, low firing thresholds ensure responses after only a few input spikes. The upper plot in Fig. 4 shows the number of unclassified examples over time, i.e. the time until the first output spike is produced, and the lower plot shows the classification error of the ConvNet using only the first output spike. A first layer threshold of 0.25 leads to very short output latencies (on the order of a few milliseconds) whereas a first layer threshold of 20 leads to greater latencies but more than 98% precision after the first spike. Note, however, that the limited precision in the case of a threshold of 0.25 is not ultimately problematic as extended execution time will yield more spikes with correct outputs.

VI. Discussion

This work presents a methodology for converting traditional neural networks to SNNs while maintaining high accuracy, and, with the introduced method of normalization, reduced evidence integration time to obtain the same accuracy. While previous investigations have examined the conversion of traditional neural networks to SNNs [16], [18], [19], we have focused here on improving the converted network by adjusting the parameters of the spiking neurons. In particular, we investigated typical sources of performance loss in SNNs and presented recipes for how to best address them.

Although the proposed model-based weight normalization led to a considerable slowdown, both the ConvNet and the FCN were still able to achieve high performance. The data-based normalization on the other hand improved latencies while assuring almost no loss due to conversion. This speedup is partly due to neurons in the deeper hidden layers being less activated. Therefore the data-based normalization actually *increases* the weights of those layers, leading to reduced latencies compared to just normalizing the first layer. This property of the data-based normalization will become especially important for converting state-of-the-art networks for challenging real-world tasks, where it is common to use more than ten hidden layers [4]. One reason for the good performance of spiking networks with the 1000 Hz input rate is that it cannot happen that inputs are only partly presented since bright pixels will fire at every single timestep. Therefore, networks which have high enough thresholds to avoid loss due to too many input spikes will show lower errors for 1000 Hz inputs. However, the longer the simulation times, the smaller is the difference due to those occasional partial inputs and for simulations of 0.5 seconds the difference is already less than 0.02%.

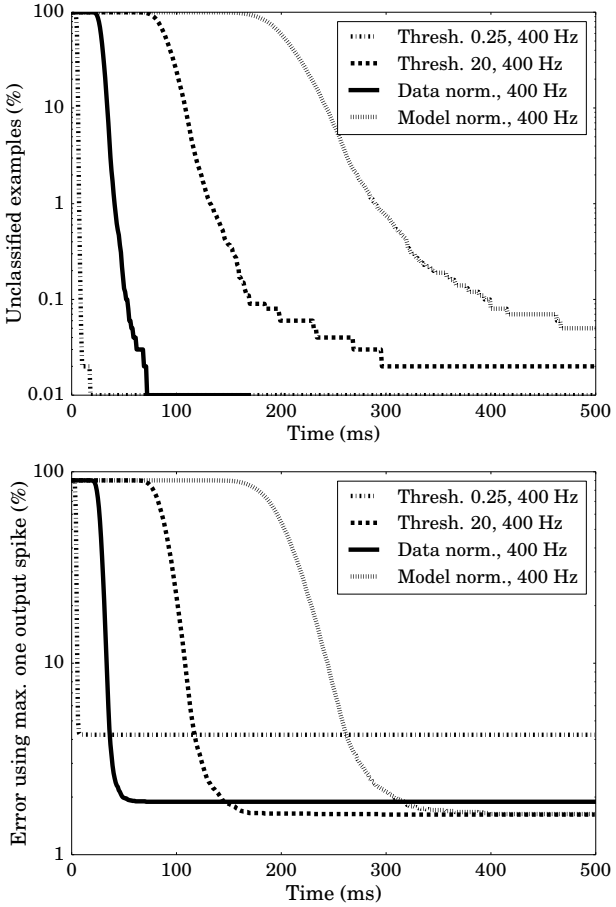

Fig. 4: Time to first output spike and performance based on the first output spike. All 10,000 MNIST test examples were presented to the spiking ConvNet for 0.5s. The upper graph shows the percentage of examples for which none of the output neurons has fired as a function of time. The lower graph shows the error rate of the network, using only the first output spike to determine the class label.

In recent years, a number of spiking architectures were developed for pattern recognition tasks. In table II we summarize their performance on the MNIST benchmark, showing that our results perform better than all previously reported SNN solutions. First implementations of spiking ConvNets focused on hardware implementations of high-speed visual object recognition from silicon retinas. The CAVIAR system [14] used a network of convolution and competition chips to process visual events and recognize objects within milliseconds, and improved convolution chips with larger kernels and lower latency were presented in [26]. The major advantage of such frame-free hardware solutions is their almost instantaneous "pseudo-simultaneous" response to input, meaning that spiking neurons in higher layer create spikes at the moment they receive inputs, rather than waiting for the rest of the neurons in lower layers to finish their computations [15]. In [18], a conversion mechanism considering also leak-currents and refractory periods is presented. Although this is more biologically realistic, the inclusion of those features renders the rate-based to spike-based network conversion more difficult.

TABLE II: Classification accuracy of different SNNs on MNIST. The results of this paper present the best reported SNN performance to date.

Network-type	Preprocessing	Performance
STDP-trained network [27]	None	93.5%
STDP-trained network [28]	None	95.0%
Spiking ConvNet [29]	Scaling, orientation detection, thresholding	91.3%
Feedward network [30]	Edge-detection	96.5%[3]
Spiking RBM [31]	Thresholding	91.9%
Spiking RBM [31]	Thresholding	92.6%
Dendritic neurons [32]	Thresholding	90.3%
Spiking RBM [22]	None	89.0%
Spiking RBM [16]	Enhanced training set	94.1%
Spiking RBM [11]	As [16], FPGA implementation	92.0%
Spiking ConvNet ([24])	None	98.3%
Spiking NN (this paper)	None	**98.6%**
Spiking ConvNet (this paper)	None	**99.1%**

Instead, here we chose to use no leak, set $v_{\mathrm{reset}} = 0$ and vary v_{thresh}, as in [19] and [24]. One of the techniques used was the clamping of v_{mem} to non-negative values, which led to a reduced performance for all simulations that we tried with these constraints.

The increasing availability of neuromorphic hardware [8], [9], [10], [11] promises an energy efficient alternative to existing pattern recognition systems based on traditional CPUs or GPUs. With the presented techniques, many of the existing state-of-the-art systems (neural networks) can be converted to SNNs and thereby harness the unique features of the neuromorphic hardware with little or no loss in classification performance.

VII. Acknowledgments

We would like to thank Olivier Bichler for helpful tips and fruitful discussions. This work was supported by the SNF Grant 200021-143337, SNF Grant 200021-146608, EU project FP7-ICT-270324, and the Samsung Advanced Institute of Technology.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012, pp. 1097–1105.

[5] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint*, vol. 312.6229, 2013.

[6] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[3] From the paper it was not clear if training and test sets were separated.

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[8] P. A. Merolla and others., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[10] S. Furber, F. Galluppi, S. Temple, and L. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[11] D. Neil and S.-C. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE Trans on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

[12] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[13] S.-C. Liu and T. Delbruck, "Neuromorphic sensory systems," *Current Opinion in Neurobiology*, vol. 20, no. 3, pp. 288–295, 2010.

[14] R. Serrano-Gotarredona and others., "CAVIAR: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing– learning–actuating system for high-speed visual object recognition and tracking," *IEEE Trans on Neural Networks*, vol. 20, no. 9, pp. 1417–1438, 2009.

[15] C. Farabet, R. Paz, J. Pérez-Carrasco, C. Zamarreño-Ramos, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, and B. Linares-Barranco, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers in Neuroscience*, vol. 6, 2012.

[16] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in Neuroscience*, vol. 7, 2013.

[17] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593–616, 2004.

[18] J. Pérez-Carrasco and others., "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets," *IEEE Trans on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013.

[19] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, pp. 1–13, 2014.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] L. Camunas-Mesa, C. Zamarreno-Ramos, A. Linares-Barranco, A. J. Acosta-Jimenez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, 2012.

[22] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, 2011, pp. 1–4.

[23] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *ICML*, 2013.

[24] D. Garbin, O. Bichler, E. Vianello, Q. Rafhay, C. Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo, "Variability-tolerant convolutional neural network for pattern recognition applications based on oxram synapses," *IEEE International Electron Devices Meeting (IEDM)*, pp. 1–13, 2014.

[25] R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," 2012.

[26] L. Camuñas-Mesa, A. Acosta-Jiménez, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32×32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 Meps throughput," *IEEE Trans on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 777–790, 2011.

[27] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Trans on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[28] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *unpublished*, 2015.

[29] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on aer motion events using cortex-like features in a spiking neural network," *IEEE Trans on Neural Networks and Learning Systems*, vol. PP, no. 99, 2014.

[30] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural Computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

[31] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," *Frontiers in Neuroscience*, vol. 7, no. 272, 2013.

[32] S. Hussain, S.-C. Liu, and A. Basu, "Improved margin multi-class classification using dendritic neurons with morphological learning," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2640–2643.