Spring 6-12-2018

An Exploration of Linear Classifiers for Unsupervised Spiking Neural Networks with Event-Driven Data

Wesley Chavez
Portland State University

An Exploration of Linear Classifiers for Unsupervised Spiking Neural Networks with Event-Driven Data

by

Wesley Chavez

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
Christof Teuscher, Chair
Melanie Mitchell
Dan Hammerstrom

Portland State University
2018

ABSTRACT

Object recognition in video has seen giant strides in accuracy improvements in the last few years, a testament to the computational capacity of deep convolutional neural networks. However, this computational capacity of software-based neural networks coincides with high power consumption compared to that of some spiking neural networks (SNNs), up to 300,000 times more energy per synaptic event in IBM's TrueNorth chip, for example [16]. SNNs are also well-suited to exploit the precise timing of event-driven image sensors, which transmit asynchronous "events" only when the luminance of a pixel changes above or below a threshold value. The combination of event-based imagers and SNNs becomes a straightforward way to achieve low power consumption in object recognition tasks. This thesis compares different linear classifiers for two low-power, hardware-friendly, spiking, unsupervised neural network architectures, SSLCA and HFirst, in response to asynchronous event-based data, and explores their ability to learn and recognize patterns from two event-based image datasets, N-MNIST and CIFAR10-DVS. By performing a grid search of important SNN and classifier hyperparameters, we also explore how to improve classification performance of these architectures. Results show that a softmax regression classifier exhibits modest accuracy gains (0.73%) over the next-best performing linear support vector machine (SVM), and considerably outperforms a single layer perceptron (by 5.28%) when classification performance is averaged over all datasets and spiking neural network architectures with varied hyperparameters. Min-max normalization of the inputs to the linear classifiers aides in classification accuracy, except in the case of the single layer perceptron

classifier. We also see the highest reported classification accuracy for spiking convolutional networks on N-MNIST and CIFAR10-DVS, increasing this accuracy from 97.77% to 97.82%, and 29.67% to 31.76%, respectively. These findings are relevant for any system employing unsupervised SNNs to extract redundant features from event-driven data for recognition.

Contents

List of Tables

List of Figures

Chapter 2

Introduction

Unsupervised neural networks, spiking and non-spiking, do not require labels in order to find statistical regularities, or features within a data set. These features, however, can be useful in the context of classification. In this case, a classifier is needed to map the combination of features that are present in a data sample to the class that the sample belongs to. The classifier must learn what combination of features is likely to be associated with each class by training the classifier with thousands of labeled data samples. After training takes place, the classifier is evaluated by holding its learned parameters fixed, and determining its classification accuracy on a testing data set. One of our goals here is to determine which classifier performs the best, as determined by highest classification accuracy in more than one task.

Also of concern in this work is low power consumption. This is why we only deal with spiking neural networks (SNNs), as they have been shown to require less energy per synaptic event (the transmission of a neuronal spike to a target neuron) than traditional software-based artificial neural networks [16]. For low power consumption, another desirable property of SNNs is sparsity, where many of the neurons are inactive at any given time. Unsupervised SNNs perform the task of extracting features from a dataset (without knowledge of a given data sample's class), where a spike corresponds to the existence of a feature in a given data sample. After many spikes occur in response to a data sample, a classifier can be trained on the spiking activity of many samples, where a per-neuron spike rate is

usually the input to the classifier. The spike rates may be linearly combined in order for the algorithm to make a "guess" as to what class the data sample belongs to. The calculation of non-linear combinations, however, is more computationally expensive, more time intensive, and not amenable to low-power applications. Also, the testing accuracy of linear classifiers rivals that of non-linear classifiers in the case of a large number of features [31], so we compare only linear classifiers for this work.

This work has five main contributions. (1) Most importantly, it compares the accuracy performance of three linear classifiers for two hardware-friendly, unsupervised SNNs, Simple Spiking Locally Competitive Algorithm (SSLCA) and HFirst, on two event-driven image recognition datasets. SSLCA and HFirst are chosen because they are representative of common spiking neural network models, and the simulation codes are publicly available. No classifier comparisons have been performed for this nature of data (SNN activity in response to event-based data), so here we compare accuracy of linear classifiers for thousands of experiments with different hyperparameter variations. (2) This work achieves state of the art classification accuracy for SNNs in the two event-based object recognition datasets, N-MNIST and CIFAR10-DVS, increasing classification accuracy from 97.77% to 97.82%, and 29.67% to 31.76% respectively. (3) This work explores key hyperparameters for both SSLCA and HFirst in order to determine optimal values for classification. (4) This work improves the recognition accuracy of the HFirst SNN from 71.15% on the N-MNIST dataset to 97.82%. (5) This work improves an event-driven classifier by 0.73% on the N-MNIST dataset by including a commonly-used "bias" term for softmax regression.

Chapter 3

Background and Related Work

3.1 Event-Based Imaging

Conventional frame-based image sensors are highly redundant, especially in the case of capturing static visual scenes. In the total absence of movement within the scene, each frame of data is the same as the previous frame, plus some additional noise. The transmission of this redundant data is very inefficient, which led to the development of a new type of sensor, the event-based sensor. Event-based vision sensors, or address event representation (AER) vision sensors, are in contrast to frame-based sensors, in that each pixel only transmits information when its log luminance changes above or below a certain threshold, as shown in Figure 3.1.

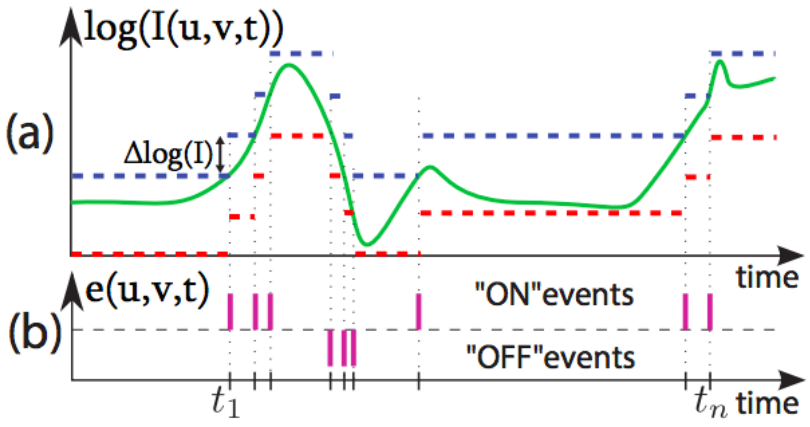

Figure 3.1: An example timing diagram for a pixel of an event-based imager. (a) The green signal is the log intensity/luminance of the input to the pixel, and the blue and red dashes are the thresholds that must be crossed in order to elicit a positive or negative event, respectively. (b) Positive and negative events in response to the signal from (a). Note: in this thesis, we only consider positive events. Figure from [20].

When this happens, the pixel in question sends an "event" on an arbitrated asynchronous bus. An event consists of a digital representation of both the pixel's x and y coordinates (or addresses) in the sensor array, and the event's polarity (whether the luminance increased or decreased). Each pixel in an event-based sensor is asynchronous and independent of the other pixels. Advantages over frame-based sensors are: reduced redundancy, reduced latency (on the order of microseconds), and increased dynamic range [3]. Biology is another inspiration for the development of these types of imagers. Like the retina, event-based imagers asynchronously produce output events (neural spikes).

3.2 N-MNIST

One of the most widely-used datasets for image recognition is the MNIST database of handwritten digits, which is a set of 70,000 labeled images (60,000 training and 10,000 testing) of 28 x 28 pixel handwritten digits, 0-9 [12]. Example MNIST patterns are depicted in Figure 3.2.

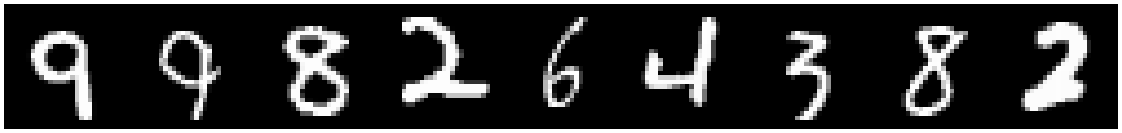

Figure 3.2: Some digits from the MNIST dataset

With a lack of event-driven image recognition datasets, Orchard et al. [19] introduced N-MNIST (Neuromorphic MNIST), a conversion of MNIST to event-based data. This dataset is converted in order to avoid non-realistic interpolations of frame-based video. Since MNIST is a dataset of images, and static images do not produce events when being displayed to an event-driven camera, Orchard et al. used a pan-tilt platform to move an Asynchronous Time-based Image Sensor

(ATIS) [22] in a systematic way while recording all 70,000 handwritten digits being displayed on a monitor in sequential fashion. This consists of three movements, or saccades, of each digit that form an isosceles triangle. This produces a dataset of 70,000 examples of 34 x 34 pixel event streams. Figure 3.3 shows example patterns for this dataset, depicted as a sum of events for the duration of each recording.

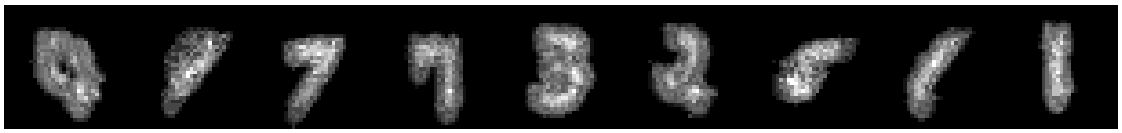

Figure 3.3: Some example digits from the N-MNIST dataset, shown as an sum of events per pixel.

3.3 CIFAR10-DVS

The CIFAR-10 dataset [11] is another popular computer vision dataset, and is composed of tiny color images (28 x 28 pixels). The 10 object classes in the dataset include *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. A few examples of CIFAR-10 patterns are shown in Figure 3.4.

Figure 3.4: Some example images from the CIFAR-10 dataset

CIFAR-10 is considerably more difficult than MNIST. For comparison, at the time of this writing, current state of the art convolutional neural networks achieve 99.79% classification accuracy on MNIST [28], and 97.69% on CIFAR-10 [30].

Li et al. [13] created its event-based counterpart, CIFAR10-DVS, as a more challenging event-based object classification task, as the color information in each image is lost (event-based imagers only respond to luminance changes). Li et

al. report a highest benchmark classification accuracy of 29.67%. In a similar manner to the creation of N-MNIST, CIFAR10-DVS was generated with closed-loop movements of an ATIS [15] presented with 10,000 of the CIFAR-10 images 1,000 of each object class. The ATIS that Li et al. used produces 128 x 128 pixel event streams, which is downsampled in this thesis to 32 x 32 pixels. Figure 3.5 shows example CIFAR10-DVS patterns as a per-pixel sum of events for each recording.

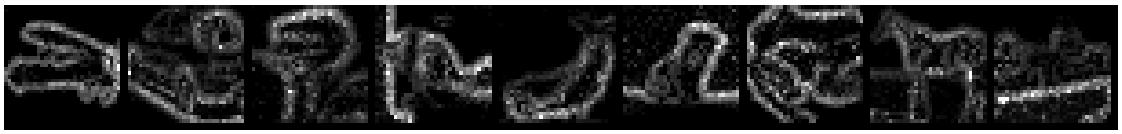

Figure 3.5: Some example images from the CIFAR10-DVS dataset, shown as a sum of events per pixel

3.4 Sparse Coding / LCA

Sparse coding (approximation) is an unsupervised generative algorithm, one that attempts to learn statistical regularities, or basis functions, within a multidimensional data set (images in this case), along with their corresponding coefficients, or neuronal activations [18]. These activations indicate how prevalent a given basis function is in the current image. These basis functions are often handcrafted, and are usually oriented edges (Gabor filters) in the case of classification of natural images. In an optimal sparse approximation, few of the activations should be non-zero. One reason for this is the observation that natural images can generally be described with a small number of basis functions, and enforcing sparse activations of the basis functions required to represent/reconstruct an image ensures that basis functions that are not as helpful in describing the image will be suppressed, making for a more efficient approximation. Locally Competitive Algorithm (LCA)

is a sparse coding algorithm that employs lateral inhibition between neurons in a layer of a sparse neural network to force these neurons to "compete" for representation of an input image [25]. These neurons act as leaky integrate-and-fire neurons, charging up from a membrane potential $u_m(t)$ of zero, and firing when this potential exceeds a certain threshold value. Only active neurons (ones above the threshold) inhibit others in an attempt to represent the current image. In sparse coding of a time-varying image $I(t)$, given a set of basis functions ϕ, LCA computes thresholded neuronal activations $a_m(t) = T(u_m(t))$ to give an approximation of the image $\hat{I}(t)$:

$$\hat{I}(t) = \sum_m a_m(t)\phi_m \tag{3.1}$$

In order to compute these activations and produce a "good" approximation, or reconstruction, of the image, the algorithm needs to iteratively minimize both the mean squared error between $\hat{I}(t)$ and $I(t)$, and the number of active neurons:

$$E(t) = \frac{1}{2}\left\|I(t) - \hat{I}(t)\right\|^2 + \lambda \sum_m C(a_m(t)) \tag{3.2}$$

The regularizer's cost function $C(a_m(t))$ is based on the coefficient threshold function, and λ is a tradeoff parameter between the desire to accurately reconstruct an input and the desire to remain sparse. In order to minimize equation 3.2, a steady state of neuronal activations must be reached, corresponding to $\frac{\delta E(t)}{\delta a_m(t)} = 0$. Rozell et al. derives neuronal dynamics as a non-linear ordinary differential equation:

$$\dot{u}_m(t) = \frac{1}{\tau}\left[b_m(t) - u_m(t) - \sum_{n \neq m} G_{m,n} a_n(t)\right] \tag{3.3}$$

where $b_m(t)$ is the inner product between $I(t)$ and a given basis function ϕ_m,

and is proportional to how well a basis function matches its input. In order to promote competition between neurons, active neurons inhibit other neurons with an intensity proportional to both how active the neuron is, as well as the similarity of their basis functions. This similarity $G_{m,n}$ is given as the dot product between ϕ_m and ϕ_n. This inhibition allows "stronger" neurons to prevent "weaker" ones from firing, and prevents linearly dependent sets of basis functions from being active.

3.5 Simple Spiking Locally Competitive Algorithm (SSLCA)

SSLCA implements the LCA in hardware realized as a memristor crossbar network, with the goal being low power consumption [29]. In the network, simplified in Figure 3.6, input spikes are converted to voltages that are applied to rows of the crossbar.

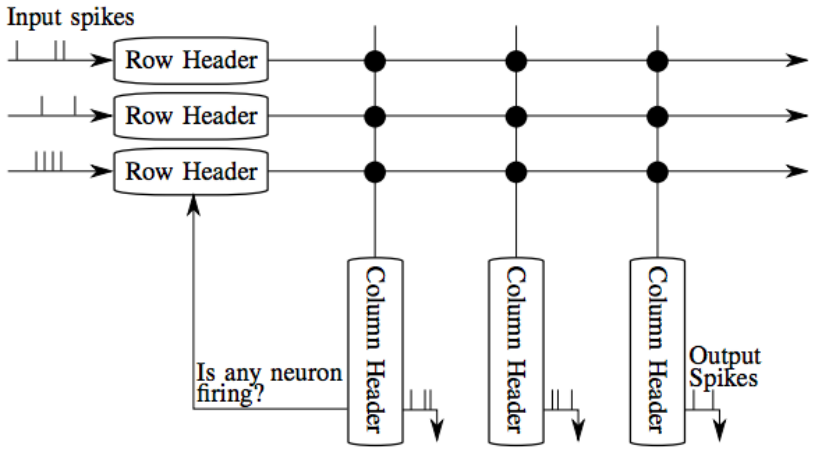

Figure 3.6: A simplified depiction of the SSLCA architecture. Input spikes produce current on a crossbar row. Inputs that are more similar to a basis function (the learned resistances in a column) will produce more current through a column wire corresponding to that basis function. When an output neuron spikes, row headers are inhibited. Figure from [29].

Memristors connect rows to columns, and the pattern of their resistances in each column corresponds to a basis function. This allows a dot product computation in each column that corresponds to $b_m(t)$ in equation 3.3, as the current in each column is proportional to how well the input matches its basis function. At the header of every column is a capacitor that acts as a leaky integrator. When the voltage across this capacitor reaches a given threshold, the column header generates a spike. When this happens, current flows backwards through the network to charge inhibitory capacitors in the row headers. In order to enforce sparsity, when a column header generates a spike, all column header voltages are reset to 0. Therefore, columns cannot spike simultaneously. This is one difference between SSLCA and LCA. However, this does not mean that only one basis function is able to represent a given input. The network is stochastic due to the specific time of arrival of the input spikes, and multiple spikes can be collected by presenting the network with input spikes corresponding to the same input multiple times. This allows flexibility in the architecture, as the sparsity level can be specified, but lacks in speed of processing.

In order to learn basis functions that are suitable for representing input images, the crossbar network is trained with input patterns x, and updates its basis functions using Oja's rule [17], which modifies a weight proportionally to the reconstruction error multiplied by the neuronal activation $\Delta w_{i,m} = \eta a_m r_i$, where $r_i = x_i - \sum_m \phi_{i,m} a_m$. η is the learning rate, and training is performed until suitable basis functions have been learned.

SSLCA does not consume much power. In the case of a 8×8 input with 128 SSLCA neurons, with the parameter $Rf_{avg} = 0.35$, SSLCA consumes about 30mW. In this work, we use a 32×32 input array, and we use 100 neurons for

most of the experiments.

One thing to note about the SSLCA implementation is that it substitutes input spikes with a Poisson-distributed spike stream. This means that the algorithm does not care about exact spike timing within the window of input presentation, only about the spike rate of each individual input pixel.

3.6 HFirst

In contrast with SSLCA, one spiking model extensively relies on information carried by the timing of a spike. HFirst [20] is a hierarchichal spiking neural network architecture intended for object recognition in event-based data designed by Orchard et al. It is a digital model that has been implemented in FPGA at a low computational cost, with minimal power consumption. Like SSLCA, each neuron in the model is implemented as a leaky integrate-and-fire (LIF) neuron, but with linear decay. When a HFirst neuron spikes, its weight is added to the membrane potential of the neuron in the next layer that it is connected to. When the membrane potential of a neuron in HFirst reaches a set threshold voltage V_{thresh}, it too sends a spike downstream to the next layer in the network. HFirst is also similar to SSLCA in that when a neuron spikes, lateral connections cause all neuron membrane potentials in the same position to reset to zero. Unlike SSLCA, after a neuron spikes, a refractory period is entered, where the neuron is not allowed to spike. The HFirst architecture extensively relies on the assumption that the first neuron to spike has a maximal response to its stimulus. This is because of two observations: (1) Sharper edges in an input stimulus result in larger temporal contrast, generating events at that location before locations with less spatial frequency in the stimulus. (2) Orchard et al. empirically show that the higher

the spatial correlation between a neurons input weights and the spatial pattern of incoming spikes, the stronger it is activated. For example, a LIF neuron that tuned to a 90 degree edge stimulus (it has weights corresponding to a vertical line) crosses a threshold before other neurons tuned to different orientations. Neurons tuned to similar orientations in the same neuronal layer (75 degrees, 105 degrees) are also strongly activated, but since the neuron tuned to 90 degrees spiked first, these other neurons are reset.

Orchard et al. introduce four neuron layers in the HFirst architecture: S1, C1, S2, and C2, which are depicted in Figure 3.7. S stands for simple, and C stands for complex. These layers are named after different types of neurons found in the biological primary visual cortex. S1 neurons perform orientation extraction. In the S1 layer, neurons receive input directly from the stimulus. S1 receptive fields (the region of influence by the preceding layer of neurons for each neuron in the layer in question, or the size of the weight matrix between these two layers) are 7×7 pixels, and there are 12 neurons tuned to different orientations at each pixel. In the case of N-MNIST, which has examples that are 34×34 pixels, this results in $34 \times 34 \times 12 = 13{,}872$ neurons in the S1 layer. Note: in [20], they use an S1 layer size of $128 \times 128 \times 12$, but for implementation with N-MNIST, the layers are reduced by a factor of four. The weights for neurons in the S1 layer are implemented as Gabor filters tuned to the 12 orientations in increments of 15 degrees. This is in contrast to SSLCA, which learns the weights needed to represent an input. Instead, HFirst implements what is referred to as a "bag of words" method, using hand-crafted Gabor filter values as weights. C1 receptive fields are comprised of 4×4 S1 neurons, and these receptive fields do not overlap. This results in $8 \times 8 \times 12 = 768$ C1 neurons for classifying N-MNIST examples. In most convolutional

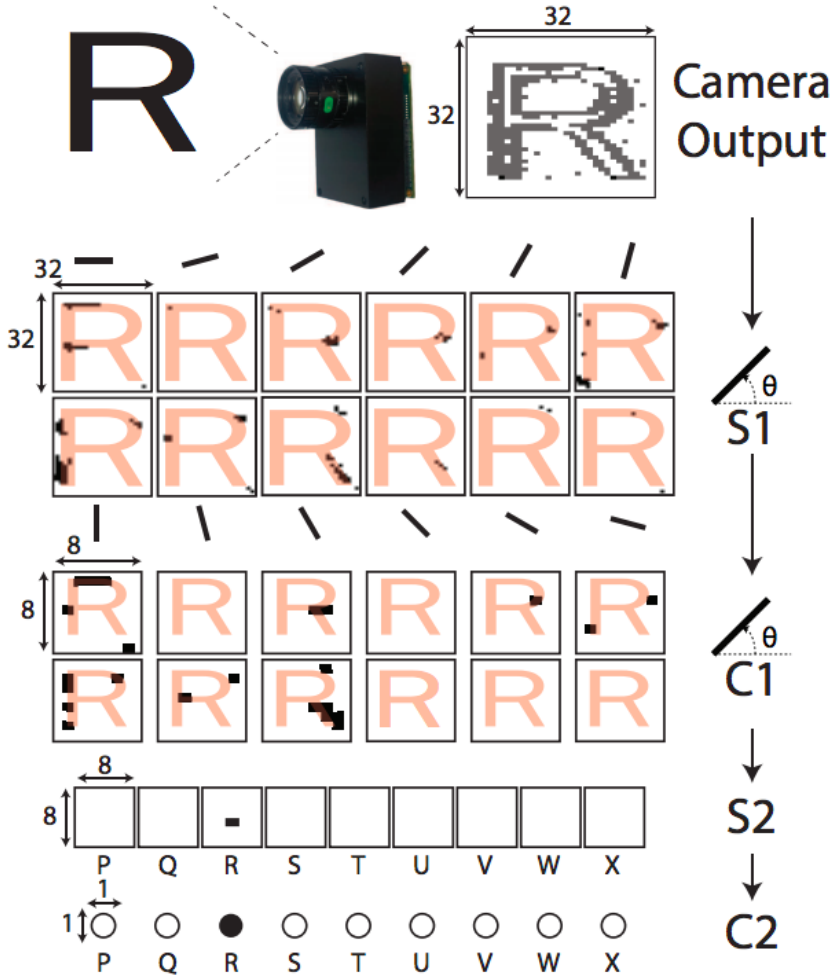

Figure 3.7: A visual description of HFirst's four neuron layers. S1 performs orientation extraction, C1 performs spatial pooling, and S2/C2 classify the example with template matching. Figure from [20].

neural networks, max pooling layers are used, in which a neuron holds the value of the maximum membrane potential in its receptive field. This introduces more non-linearity in the system (essential for complex multidimensional problems such as image recognition) and prevents the model from overfitting the training data. Neurons in the C1 layer implement a similar operation, but instead of assuming the maximum value of its receptive field, a neuron in the C1 layer determines

which S1 neuron spiked first. The threshold for C1 neurons is set very low, so that only one S1 spike is needed for a C1 neuron to spike. C1 neurons also have lateral reset connections between neurons of a different orientation at the same location, in order to implement the pooling.

S2 and C2 neurons implement HFirst's classifier as a template matcher. S2 neurons have a receptive field of $8 \times 8 \times 12$ C1 neurons. The receptive fields of S2 neurons overlap with a stride of one (one S2 neuron location for every C1 neuron location). Normally, there would be $8 \times 8 \times N_y$ S2 neurons, N_y being the number of classes in the dataset. However, since (in the case of N-MNIST) the S2 receptive field size would match the number of C1 neuron locations (8×8), only $1 \times 1 \times N_y$ S2 neurons are used to eliminate redundant computations. In order to perform template matching, weights between C1 and S2 must be learned to answer the question: How much of each C1 spike should contribute to the the hypothesis that the example stimulus carries the label of each class? Instead of learning the weights from C1 to S2, HFirst counts the spikes from each C1 neuron during the presentation of the spike trains of all examples in a training set from a particular class, and normalizes those counts to have a Euclidean norm of 100. For example, in the case of N-MNIST, HFirst first initializes a weight matrix of size $8 \times 8 \times 12 \times 10$ to zeros. HFirst then adds one to the index of every C1 neuron that spikes (one of $8 \times 8 \times 12$) and at the index of the class that the example belongs to (one of 10). The Euclidean norm of these counts is set to 100 for each of the 10 S2 neurons to use as weights. Optionally, HFirst uses a C2 layer to perform pooling of S2 neurons, just as C1 neurons pool from S1 neurons. The class that generates the most S2 (or C2) spikes is chosen as the most likely label.

For an input size of 128×128 pixels, HFirst consumes from 150 - 250 mW in

their digital implementation.

3.7 Perceptron

The perceptron [24], or single-layer perceptron, is one of the most simple forms of artificial neural networks. It is an algorithm for learning a linear, binary classifier to map an input vector x to an output binary value y (in the case of distinguishing between two classes). Specifically,

$$y = \begin{cases} 1 & w \cdot x + b > 0 \\ 0 & otherwise \end{cases} \tag{3.4}$$

where w is the weight vector, and b is the bias. If y is 1, the example is classified as a positive instance of a class. The cost function minimized by the perceptron across N training examples is the sum of squared errors:

$$J(w) = \sum_{n=1}^{N} \frac{1}{2}(desired_n - y_n)^2 \tag{3.5}$$

In order to minimize this function, the weight vector is modified in an online fashion (the weights are updated after every presentation of a training example) with the update rule for a single example:

$$w_i := w_i + (desired - y)x_i \tag{3.6}$$

If the perceptron misclassifies an example, this update moves the weights in the right direction (the direction in which the classifier is more likely to classify this example correctly the next time it is input to the perceptron). If the perceptron makes a correct prediction for an example, the network weights do not change. This

is important, as the perceptron only optimizes the network to correctly classify as many examples as possible. In essence, a single-layer perceptron modifies a hyperplane that acts as a linear decision boundary between two classes. In order to extend the perceptron to more than two classes, the perceptron uses j weight vectors, where j is the number of classes, and chooses a class according to:

$$y = \arg\max_{j}(w_j \cdot x + b_j) \tag{3.7}$$

Note that y is no longer binary. If $desired = y$, all weights are unchanged. Otherwise, the update rule becomes:

$$\begin{aligned} w_{ij} &:= w_{ij} - x_i \quad \text{for negative classes} \\ w_{ij} &:= w_{ij} + x_i \quad \text{for the positive class} \end{aligned} \tag{3.8}$$

This form of multiclass classification is known as "one vs. all", where there exists one weight vector for every class, and each hyperplane attempts to separate data from the corresponding class and data from all other classes. If all examples are able to be separated with a hyperplane (or multiple hyperplanes in the case of more than two classes), the perceptron algorithm will converge, as no more weight updates take place. Also, a learning rate is not used, as it just rescales w, and does not change y.

3.8 SVM

One problem with the perceptron algorithm is that for linearly seperable data, it does not necessarily find the optimal separating hyperplane, as the weights are no longer updated after convergence. The idea behind a linear support vector

machine (SVM) is to maximize the distance between the hyperplane and training instances of both classes. Note that in the case of two dimensions, the separating hyperplane is a line. The support vectors are the input vectors x that are closest to the hyperplane. A linear SVM defines hyperplanes such that

$$\begin{aligned} w \cdot x + b \geq 1 \text{ when desired} = 1 \\ w \cdot x + b \leq -1 \text{ when desired} = \text{-1} \end{aligned} \tag{3.9}$$

which can be combined into $desired(w \cdot x + b) \geq 1$ The distance between a point (x_0,y_0) and a line $w_1 x + w_2 y + b = 0$ is $\frac{|w_1 x_0 + w_2 y_0 + b|}{\sqrt{w_1^2 + w_2^2}}$; consequently the distance between a support vector and the hyperplane is $\frac{|w \cdot x + b|}{\|w\|} = \frac{1}{\|w\|}$, where $\|w\|$ is the Euclidean length of w. In order to maximize this distance between support vectors from both classes and the hyperplane, we therefore need to minimize $\|w\|$, with the condition that there are no input vectors within the margins. However, when the data is not linearly seperable, as in many image classification tasks, we cannot satisfy this condition. Thus we introduce the hinge loss function

$$\max(0, 1 - desired(w \cdot x + b)) \tag{3.10}$$

This function is zero if x is on the correct side of the margin, and is proportional to the distance from x to the margin otherwise. SVMs solve the unconstrained optimization problem:

$$\min_w \frac{1}{2} w^2 + C \sum_{n=1}^{N} \xi(w; x_n, desired_n) \tag{3.11}$$

where ξ is the hinge loss function of equation 3.10 in the case of L1-loss SVM, and $\max(0, 1 - desired(w \cdot x + b))^2$ for L2-loss SVM. The term $\frac{1}{2} w^2$ in equation

3.11 can be replaced with $\sum_j |w_j|$ in the case of L1-regularized SVMs. The parameter C is a tradeoff parameter that indicates how important it is to classify examples correctly, while sacrificing the ability of the SVM to keep a larger margin between the support vectors and the hyperplane. C is in the range zero to infinity. The minimization problem in equation 3.11 can be solved by reducing it to the Lagrangian dual of a constrained optimization problem, efficiently solvable by convex quadratic programming algorithms. An open-source library for large-scale linear classification called LIBLINEAR [5] uses a technique called stochastic dual coordinate descent (SDCD) to iteratively (example-by-example) solve this optimization problem until a near-optimal solution is obtained. LIBLINEAR includes eight solvers designated here as s that minimize slightly different problems. A description of each solver is outlined in Table 3.1.

s	Solver Type
0	L2-regularized logistic regression (primal)
1	L2-regularized L2-loss support vector classification (dual)
2	L2-regularized L2-loss support vector classification (primal)
3	L2-regularized L1-loss support vector classification (dual)
4	Support vector classification by Crammer and Singer [2]
5	L1-regularized L2-loss support vector classification
6	L1-regularized logistic regression
7	L2-regularized logistic regression (dual)

Table 3.1: Types of solvers included in the LIBLINEAR package

We treat these eight solvers as separate classifiers for comparison in this work. We also use the linear classifier provided by another open-source library, LIBSVM [1]. This classifier is a L2-regularized, L1-loss support vector classifier similar to $s = 3$ in Table 3.1, with one distinct difference: LIBSVM uses the "one-vs-one" strategy in regards to multiclass classification, while the LIBLINEAR solvers use "one-vs-all". Consider the case of a classifier attempting to distinguish between

N classes. In one-vs-all classification, there are N binary classifiers, each classifier trying to determine if the current pattern is a positive instance of class n. In one-vs-one classification, there are $\frac{N^2-N}{2}$ classifiers, each distinguishing between two of the N classes. Each classifier votes for the most likely class, and the class with the most votes is chosen as the winner. It is not clear whether one-vs-one or one-vs-all is more suitable for higher classification accuracy, so we compare the LIBSVM classifier to the LIBLINEAR classifier, with $s = 3$.

3.9 Logistic/Softmax Regression

Logistic regression is an algorithm that, like the perceptron, attempts to map an input vector x to an output y, parameterized by weights w. Instead of the Heaviside step function of the perceptron, equation 3.4, that outputs a binary decision, logistic regression uses the sigmoid function as its activation function:

$$y = \frac{1}{1 + e^{-w \cdot x + b}} \tag{3.12}$$

It still maps the input from 0 to 1, but y in this case is interpreted as a probability that the current example is a positive instance of a class, that is:

$$y = P(class = 1|x; w) = 1 - P(class = 0|x; w) \tag{3.13}$$

The sum of squared errors, equation 3.5, is not an appropriate cost function for logistic regression because it is not convex for the sigmoid activation function. Logistic regression uses the cost function:

$$J(w) = \sum_{n=1}^{N} (desired_n - 1) \log(1 - y_n) - desired_n \log(y_n) \tag{3.14}$$

where N is the number of training examples and $desired_n$ is either 0 or 1. This ensures a low cost if y_n is near $desired_n$, and a high cost otherwise. In order to minimize this cost, the weight update rule for a single example is:

$$w_i := w_i + \eta(desired - y)x_i \tag{3.15}$$

where η is the learning rate. If y is near $desired$, this gives a very small update. Softmax regression is the extension of logistic regression to more than two mutually exclusive classes. Instead of equation 3.13, softmax regression attempts to estimate the probabilities for j classes and replaces the sigmoid function with a softmax function:

$$y_j = P(class = j|x; w) = \frac{e^{w_j \cdot x + b_j}}{\sum_j e^{w_j \cdot x + b_j}} \tag{3.16}$$

This function represents a probability distribution across classes and ensures that the computed probabilities add up to 1. The cost function in this case is:

$$J(w) = -\sum_{n=1}^{N} \sum_j desired_{nj} \log y_{nj} \tag{3.17}$$

where $desired_{nj}$ is again a binary value, 0 for negative classes and 1 for the positive class. In order to decrease this function, the update rule is as follows:

$$w_{ij} := w_{ij} + \alpha(desired_j - y_j)x_i \tag{3.18}$$

Gradient descent in this way is performed until "suitable" weights have been learned.

3.10 Related Work

In [26], Stromatias et al. propose a novel method for training an event-driven classifier that achieves 97.77% accuracy on the N-MNIST dataset, the highest accuracy reported to date with a convolutional spiking neural network. In addition, they also retrained the output layer of a different feature-extracting spiking neural network, which increased its performance by 2% on a dataset. Their approach involves treating the normalized histograms of per-neuron spike counts as inputs to a softmax classifier. Since we want to improve on this classifier, we follow the same training scheme for my experiments, the details of which are explained in Section 4.4. The classifier of Stromatias et al. also distinguishes classifier outputs y with and without membrane potential leakage. In this thesis, we only consider non-leaky outputs y for all classifiers.

There have been attempts to compare classification accuracy on various datasets using linear classifiers, although none have dealt with spiking, event-based data. Tang [27] compared an L2-loss, L2-regularized, linear SVM (s=1 in Table 3.1) to a softmax classifier, both as the final layer of a deep convolutional neural network for three datasets: a face recognition dataset, MNIST, and CIFAR-10. The SVM outperformed the softmax classifier by 0.1%, 0.12%, and 2.1%, respectively. In [32], Zeiler and Fergus similarly placed linear classifiers on top of a deep convolutional model to classify images from two larger datasets, where the linear SVM achieved 0.1% higher classification accuracy than the softmax classifier in one dataset and 0.9% worse in another dataset. Do et al. [4] applied classifiers to features important for text classification, where a softmax classifier obtained an average of 9.23% higher classification accuracy than that of a nonlinear one-vs-all SVM classifier on four text classification tasks. This is surprising, because nonlinear SVMs by

definition should be able to model data at least as well as SVMs with a linear kernel. It is apparent that hyperparameters need to be tightly controlled in order to determine which classifier is "better".

Similar to the softmax vs. linear SVM problem, it is not conclusive whether, in the case of linear SVMs, a "one-vs-one" strategy performs better than a "one-vs-all" strategy. Conflicting studies have been done that argue for either strategy ([2], [23]), with neither method proving a clear winner. This is why we look at the performance of the linear SVM from the LIBSVM package, which uses a one-vs-one strategy, as well as the eight classifiers from the LIBLINEAR package, which use the "one-vs-all" strategy.

Chapter 4

Methods

In order to gather statistics for accuracy comparisons, both SSLCA and HFirst were presented with spiking samples from either N-MNIST or CIFAR10-DVS datasets to train w, then the testing examples are presented to determine the model's accuracy, i.e., what percent of testing examples it correctly classified. For N-MNIST, we used the same training/testing sets of both [12] and [19] of 60,000 examples and 10,000 examples, respectively. In the case of CIFAR10-DVS, we used the first 800 examples from each class as training examples, and the last 200 as testing examples for a total of 8,000 training and 2,000 testing. In both datasets, we shuffled the examples of the training set with the same random seed for all experiments. In the creation of CIFAR10-DVS, Li et al. recorded six loops of four saccades by the ATIS. In presentation of this dataset to SSLCA and HFirst, we only used spikes from the first 300ms, as this further reduces data redundancy. Also, we downsampled the dataset by four in both the x and y dimensions, generating 32 x 32 pixel event streams. For all experiments, only positive-polarity spikes were used. In order to perform accuracy comparisons in the events where each spiking network was not performing optimally, e.g. not enough spikes output for reasonable detection accuracy, not enough training iterations through the dataset (epochs), etc., we performed a grid search (an exhaustive sweep) of model hyperparameters for both SSLCA and HFirst. This grid search accomplishes two things. It allows us to determine the SNN hyperparameters that are more amenable to

higher classification accuracy, and also provides more data with which to compare classifiers. Unless otherwise noted, each accuracy value reported in Chapter 4 is the mean of classification accuracies for the SNN models with different hyperparameter values, as opposed to the maximum classification accuracy. This is because we would like to know which classifier performs the best, without regard to specific SNN hyperparameters.

4.1 Training SSLCA

The four most important hyperparameters for SSLCA are input duty cycle bias, number of expected output spikes, training epochs, and number of neurons. Woods et al. [29] found that increasing the duty cycle of the input spikes from $K_{max}k_{input}$ to $K_{max}(bias + (1 - bias)k_{input})$ results in higher classification accuray on the MNIST dataset, from 77% to 84%. We therefore treat $bias$ as an essential hyperparameter for exploration. We perform a sweep of $bias$ from 0.1 to 0.65 in increments of 0.05. Also, SSLCA derives important architecture parameters such as capacitance and firing voltage based on the expected number of output spikes for each example presentation. Note that this is only an average, and the actual number of SSLCA output spikes varies with each spike train presentation. With the SSLCA hyperparameter grid search, we use values of 10, 20, and 30 for the expected number of spikes. We find that training the dataset for more epochs produces a much more refined basis set, as will be discussed in section 5.1.2. We vary the number of training epochs with 1, 5, and 10 iterations over the N-MNIST and CIFAR10-DVS training sets. We also hypothesize that the number of SSLCA output neurons also affects performance, as more computational capacity should result in higher classification performance. In this experiment, we use 50, 100, and

768 neurons along with an SLP classifier. After the network has been trained, the basis functions are kept fixed for the recording of neuronal activations in response to both the training set and the testing set. These recordings will be the input to our three classifiers.

Woods et al. hint that 0.35 is an optimal value for the SSLCA hyperparameter Rf_{avg}. This value is used in this thesis as well.

4.2 HFirst

A discussion of the optimization/exploration of important model hyperparameters for HFirst has not been provided by the authors, so we varied the default settings of three hyperparameters in a grid search: threshold voltage, refractory period, and decay rate of the neurons. In their N-MNIST implementation, Orchard et al. use a default threshold voltage V_{thresh} of 150 mV for both the S1 and S2 layers. In this thesis, we experiment with V_{thresh} values of 100 mV, 150 mV, and 200 mV. For S1, C1, S2, and C2 layers, the authors use a refractory period of 5ms. In my experiments, we use values of 1ms, 5ms, and 10ms for all four layers. Last, the authors use an S1 decay rate of 25 mV/ms, whereas we use values of 20 mV/ms, 25 mV/ms, and 30 mV/ms.

We run HFirst on both datasets with these 27 hyperparameter variations in order to determine classification accuracies for the HFirst architecture, but it is important to note that we exclude HFirst's template-matching classifier (layers S2 and C2), and instead substitute one of the linear classifiers used in this thesis. The C1 recordings of both training sets and testing sets are used as inputs to the classifiers in order to determine how useful the chosen Gabor functions are to classification, as well as to compare the classifiers.

4.3 Perceptron

The perceptron we used in these experiments is from the scikit-learn library [21]. Assuming the training dataset is not linearly seperable, the perceptron weights will not converge to a solution. Given this assumption, we train the networks for 100 epochs and test with the current network weights every 10 epochs for each SSLCA or HFirst implementation.

4.4 Softmax

We use the same training details as Stromatias et al. [26] for the softmax classifier. (1) We normalize the input spike counts with respect to the maximum value, resulting in values between zero and one. (We also test different normalization schemes, detailed in Section 4.6.) (2) In order to reduce the variance of the softmax algorithm and converge to a minimum in the cost function earlier, mini-batches of examples can be evaluated before computing weight updates [9] [10] [14]. This is known as mini-batch stochastic gradient descent. Instead of the stochastic gradient descent weight update rule (equation 3.18), which updates the weights in response to a single training example i, mini-batch gradient descent use the update rule:

$$w_{ij} := w_{ij} + \alpha \frac{1}{D} \sum_{i=1}^{D} (desired_j - y_j) x_i \tag{4.1}$$

and updates the weights in response to the mini-batch size D. Stromatias et al. use a mini-batch size of 500 examples training for 1,500 epochs with a learning rate α of 0.1. We use the same values in the softmax classifier experiments. One thing to note is that Stromatias et al. does not use a softmax bias b in their classifier. We observe the effect that this omission has on the classifier's accuracy. The classifier

is written in Python.

4.5 SVM

The linear SVM experiments are run with LIBSVM and LIBLINEAR. Note that s = 0, 6, and 7 from Table 3.1 are actually logistic regression classifiers, but they still have the tradeoff hyperparameter C that balances training accuracy and the ability to maintain a larger margin. Choosing different values for C affects classification performance, and is in the range of zero to infinity. We choose values of 1, 5, 10, 50, 100, and 500 for C in all experiments.

4.6 Normalization of Classifier Inputs

For the perceptron and the softmax classifier, it is not clear whether the normalization of the inputs to the classifier (per-neuron spike counts) aides in the classification task. A widely-used way to scale vectors for input to neural networks is to standardize them, that is subtracting the mean of the vector and dividing by its standard deviation:

$$x_{standardized} = \frac{x - \bar{x}}{\sigma} \tag{4.2}$$

Usually this is done for each feature in the input. However, low power consumption being an objective of SNNs, sparsity is an attractive trait. Whether these classifiers are implemented in a digital or analog network, sparse activations will cause the system to consume less power. See Figure 4.1 for the histogram of per-example spike counts of the HFirst C1 neurons in response to the CIFAR10-DVS dataset. In order to keep these sparse, we scale the input values to the range $[0, 1]$

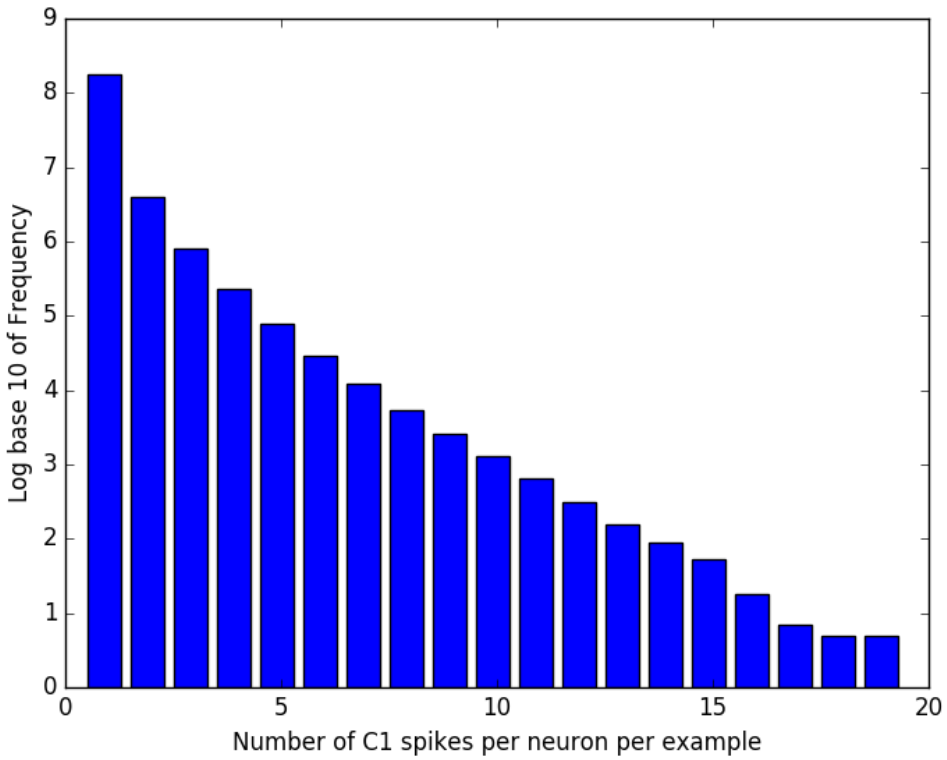

Figure 4.1: Histogram of the non-scaled inputs to the classifiers for HFirst in response to the CIFAR10-DVS dataset. Note that the y axis is the log base 10 of the frequency of C1 spike counts.

using min-max scaling. With this approach, a scaled vector x_{scaled} is computed as

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)} \tag{4.3}$$

Two normalization techniques exist: feature-dimension scaling and pattern-dimension scaling. With a matrix $x_{n,i}$ of the training data set with N training examples (patterns) and I neurons (features), feature scaling normalizes each feature in the matrix separately, and pattern scaling normalizes each pattern in the matrix separately.

Pattern-dimension scaling may help gradient descent. Take into account the image I that we are trying to classify. The image $I + 0.5$ contains the same object, yet a linear classifier will obtain a different result. Pattern-dimension scaling ensures that the image $I + 0.5$ will produce the same classification result as the image I. Pattern-dimension scaling is what Stromatias et al. [26] uses for their softmax-based event-driven classifier.

Normalization of input vectors in the feature dimension may also improve stochastic gradient descent techniques, because features that repeatedly have large values compared to other features tend to have the largest weight updates, and become most of the driving force of the algorithm's attempt to find a solution. The normalization of features ensures that this doesn't happen, and gradient descent converges faster [8].

For SVMs, Hsu et al. [7] stress the importance of min-max feature scaling, and using the same feature-wise $min(x)$ and $max(x)$ from the training set to compute x_{scaled} for the testing set. Hsu et al. note that this improves SVM accuracy from 69.2% to 89.4% in one classification task, as opposed to normalizing the testing set with $min(x)$ and $max(x)$ computed from the testing set. We use the same technique as Hsu et al. for our feature-dimension scaling experiments.

In this thesis, we explore recognition accuracies for all the linear classifiers using min-max feature scaling, min-max pattern scaling, and no scaling of the inputs.

Chapter 5

Results and Discussion

5.1 Hyperparameter Exploration for SSLCA

5.1.1 Results

In order to determine the SSLCA settings that achieve the highest accuracy on the N-MNIST dataset, we look at the case of the SLP with non-normalized inputs, and do a grid search of four hyperparameters: SSLCA bias, number of spikes, number of training epochs, and number of neurons. We average the accuracies for all 3,780 tests (14 bias values, 3 number of spikes values, 3 training epoch values, 3 number of neurons values, and 10 number of iterations values) across each hyperparameter, the results of which are in Table 5.1, Table 5.2, Table 5.3, and Figure 5.1.

Number of Spikes	Mean SLP Accuracy
10	69.18%
20	73.62%
30	74.94%

Table 5.1: Mean SLP accuracy for SSLCA on N-MNIST with no normalization for different number of spikes. More output spikes per input example leads to better classification.

Number of Training Epochs	Mean SLP Accuracy
1	65.93%
5	75.22%
10	76.59%

Table 5.2: Mean SLP accuracy for SSLCA on N-MNIST with no normalization for different amounts of training epochs. More training epochs gives the model more time to produce a basis set that is more amenable to higher classification accuracy.

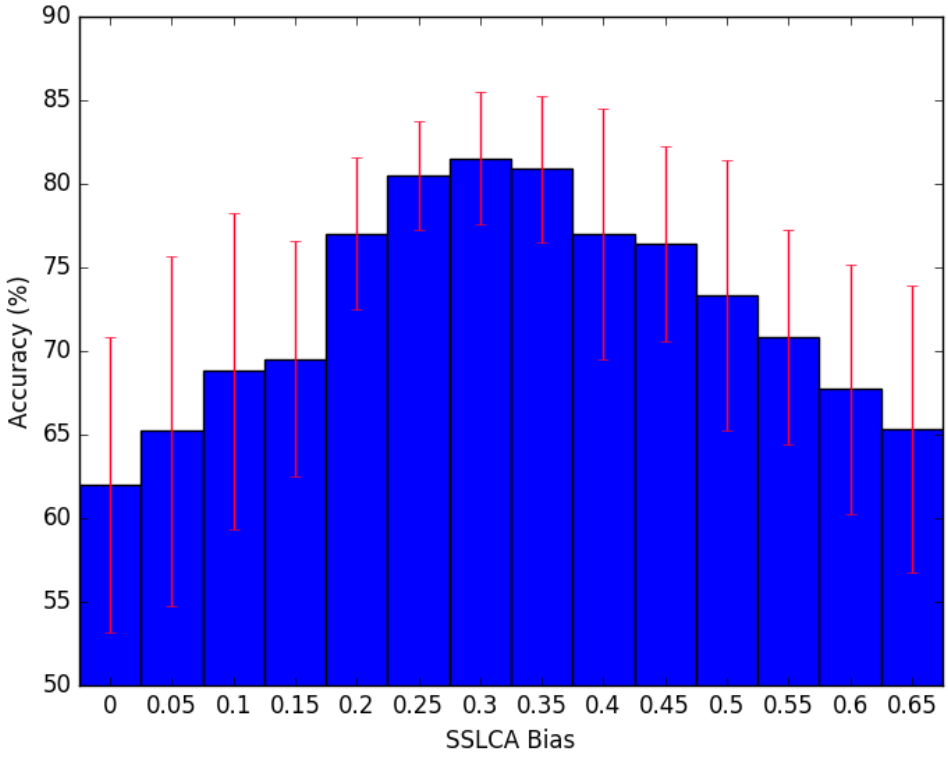

Figure 5.1: Mean SLP accuracy for SSLCA on N-MNIST with no normalization. Error bars indicate the standard deviation of classification results across SSLCA hyperparameter variations. An SSLCA bias of 0.3 causes SSLCA to produce the best basis functions for classification.

Number of Neurons	Mean SLP Accuracy	Mean SLP Accuracy at Bias 0.3
50	71.81%	77.93%
100	74.44%	81.96%
768	71.49%	84.67%

Table 5.3: Mean SLP accuracy for SSLCA on N-MNIST with no normalization for different numbers of neurons/basis functions. For some reason, low biases caused SSLCA to perform poorly with 768 neurons, hinting at a lack of stability for this combination of SSLCA hyperparameters. At at bias of 0.3, the addition of neurons aides in classification.

5.1.2 Discussion

In the case of SSLCA, recognition accuracy increases with more spikes and training epochs, as seen in Table 5.1 and Table 5.2. Also, accuracy increases with number

of neurons in Table 5.3, assuming a bias of 0.3. With the case of 768 neurons, with a bias of 0.15 and under, training for only one epoch, classification suffered a great loss in accuracy, leading to the average of accuracy for 768 neurons being so low.

It is apparent in 5.1 how much hyperparameter values matter with the SSLCA architecture. Figure 5.2 shows an example of learned weight vectors (basis functions) for N-MNIST with an SSLCA bias of 0, 1 epoch, and 10 spikes, while 5.3 shows weights for a bias of 0.3, 10 epochs, and 30 spikes.

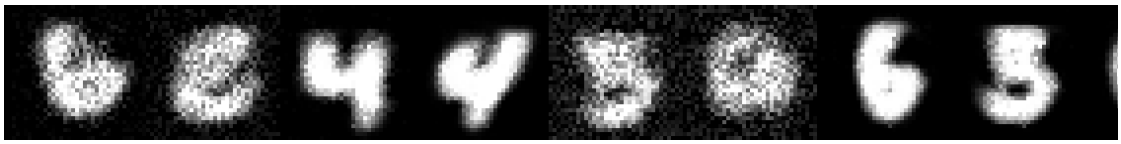

Figure 5.2: SSLCA learned weight vectors on the N-MNIST dataset, bias 0, 1 epoch, 10 spikes

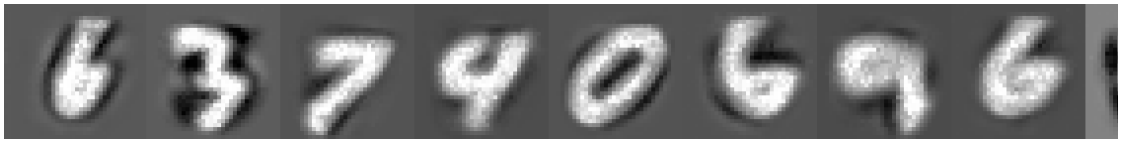

Figure 5.3: SSLCA learned weight vectors on the N-MNIST dataset, bias 0.3, 10 epochs, 30 spikes

It seems that a "crisp-looking" set of basis functions in a feature extraction model such as SSLCA performs better than a set of features with more spatial noise, such as the subsets of learned features depicted in Figure 5.2 and Figure 5.3. This is another confirmation of SSLCA's sensitivity to hyperparameter variations.

5.2 Hyperparameter Exploration for HFirst

5.2.1 Results

We perform the same tests with the SLP (no normalization) classifying HFirst activations produced in response to N-MNIST events. The hyperparameters in

this implementation are V_{thresh}, refractory period, and S1 decay rate. There are 270 total tests that are averaged in the same manner as in SSLCA, with different hyperparameters of course. Results are shown in Table 5.4, Table 5.5, and Table 5.6.

V_{thresh} (mV)	Mean SLP Accuracy
100	95.72%
150	95.30%
200	94.77%

Table 5.4: Mean SLP accuracy for HFirst on N-MNIST with no normalization for varied V_{thresh}. A lower V_{thresh} performs slightly better.

Refractory Period (ms)	Mean SLP Accuracy
1	95.39%
5	95.32%
10	95.07%

Table 5.5: Mean SLP accuracy for HFirst on N-MNIST with no normalization for different refractory period values. A smaller refractory period seems to aid in classification.

S1 Decay Rate (mV/ms)	Mean SLP Accuracy
20	95.67%
25	95.36%
30	94.76%

Table 5.6: Mean SLP accuracy for HFirst on N-MNIST with no normalization for varied S1 decay rates. A lower decay rate causes the SLP to classify better.

5.2.2 Discussion

As you can see, classification performance was not as sensitive to these HFirst hyperparameter values as it was for SSLCA hyperparameter values. However, there is a slight trend toward better classification with a smaller V_{thresh}, refractory

period, and S1 decay rate. The decrease of these values leads to more C1 spikes. The same trend is seen with the number of spikes of SSLCA neurons, although to a smaller degree.

Of course, HFirst performed feature extraction superior to that of SSLCA in regards to classification accuracy, but they are performing two different tasks. That is, HFirst is extracting features in a convolutional manner where the features (7 x 7 pixel Gabor filters) can be applied anywhere spatially in the input stream, while SSLCA learns features for the whole input space. This results in features that are not spatially reusable/translationally invariant, and neuron activations that convey no information about where a feature is located in the input image.

5.3 Softmax Bias

5.3.1 Results

In order to determine how much the bias term b helps when performing object classification, we performed a set of experiments with and without b. Here we focus on classifying SSLCA neuronal activity in response to the N-MNIST dataset by the softmax classifier without normalization of the neuronal activity, and with 100 SSLCA neurons. The mean accuracy of these 126 experiments (14 SSLCA bias values, 3 number of epochs values, and 3 number of spikes values) without a bias is 80.31%, and with a bias is 81.05%, a performance increase of 0.74%.

5.3.2 Discussion

The addition of a bias b when learning softmax weights proved to consistently increase classification performance. This makes sense, for in the hypothetical case of two features, the omission of this bias term equates to a decision boundary that

always passes through the origin. To clarify, remember that when there are two features/dimensions, the hyperplane that we want to separate classes is a line in the form of $w_1 * x_1 + w_2 * x_2 + b = 0$. Without b, the hyperplane must pass through the origin, limiting its classification performance in most cases. Therefore, it is surprising that the addition of the bias term only increases accuracy by 0.74% with softmax classification of SSLCA activations on the N-MNIST dataset. Nevertheless, the bias term is important, and only increases the amount of weights that need to be trained by one.

5.4 Classifier Comparison

5.4.1 Results

Recall that it is not intuitive whether or not the input to a linear classifier should be scaled, with either a pattern-dimension or feature-dimension normalization technique. The following figures depict the mean classification accuracy with either the SSLCA or HFirst architectures in response to either the N-MNIST or CIFAR10-DVS datasets.

For SSLCA paired with both N-MNIST and CIFAR10-DVS, there are a total of 1,260 experiments averaged together for each SLP data point (14 SSLCA bias values, 3 number of epochs values, 3 number of spikes values, and 10 number of SLP training iterations values), 126 experiments for each softmax data point (14 SSLCA bias values, 3 number of epochs values, and 3 number of spikes values), and 756 experiments for each SVM data point (14 SSLCA bias values, 3 number of epochs values, 3 number of spikes values, and 6 C values). The only hyperparameter that is held fixed in these experiments is number of neurons, which is 100. These mean accuracies are depicted in Figure 5.4 and Figure 5.5.

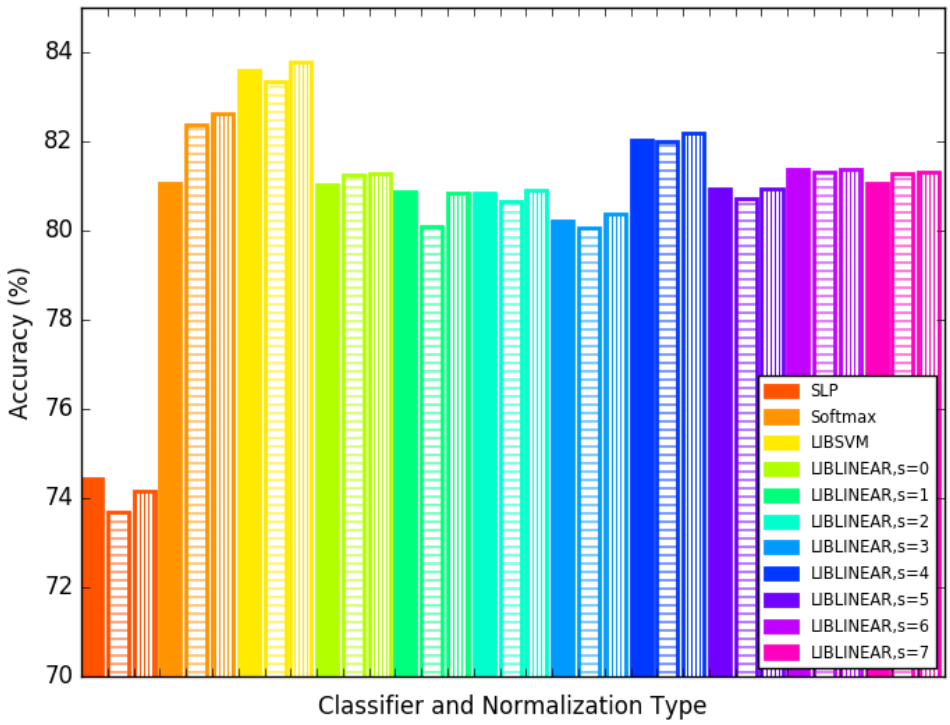

Figure 5.4: Mean classifier accuracy for the SSLCA architecture on the N-MNIST dataset. Solid bars indicate no scaling, bars with horizontal lines indicate pattern-dimension scaling, and bars with vertical lines indicate feature-dimension scaling. LIBSVM's linear SVM outperformed the rest here.

For HFirst paired with both datasets, there are a total of 270 experiments averaged together for each SLP data point (3 V_{thresh} values, 3 refractory period values, 3 S1 decay values, and 10 number of SLP training iterations values), 27 experiments for each softmax data point (3 V_{thresh} values, 3 refractory period values, and 3 S1 decay values), and 162 experiments for each SVM data point (3 V_{thresh} values, 3 refractory period values, 3 S1 decay values, and 6 C values). These mean accuracies are depicted in Figure 5.6 and Figure 5.7.

In order to determine the classifier that performed well for both SNN models paired with both datasets, we took a mean of each classifier's accuracy across the

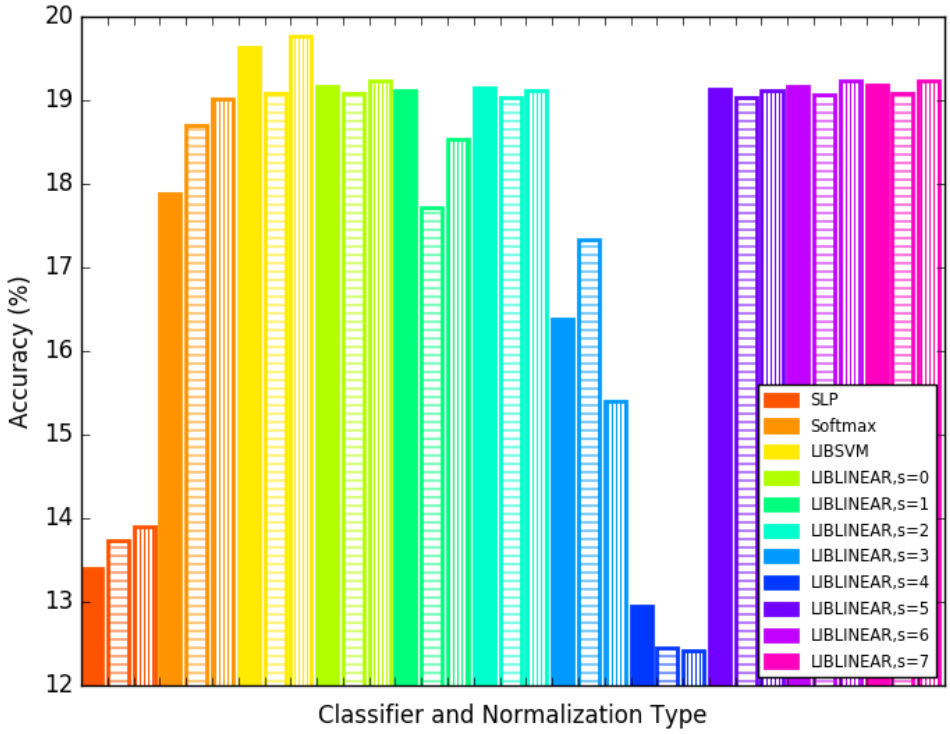

Figure 5.5: Mean classifier accuracy for the SSLCA architecture on the CIFAR10-DVS dataset. Solid bars indicate no scaling, bars with horizontal lines indicate pattern-dimension scaling, and bars with vertical lines indicate feature-dimension scaling. Again, LIBSVM's linear classifier outperformed the rest. In this case, the linear SVMs corresponding to $s = 3$ and $s = 4$ suffered a significant degradation in accuracy compared to the other SVMs. The SLP again did not prove to be a strong classifier.

four SNN/dataset combinations. These are plotted in Figure 5.8.

5.4.2 Discussion

We concede that the average of classification performance across datasets and SNN models may not be the best metric for comparing classifiers. This metric gives more weight to performance in more difficult classification tasks, as the variance of accuracy is greater. Nonetheless, there exists no other intuitive metric to compare

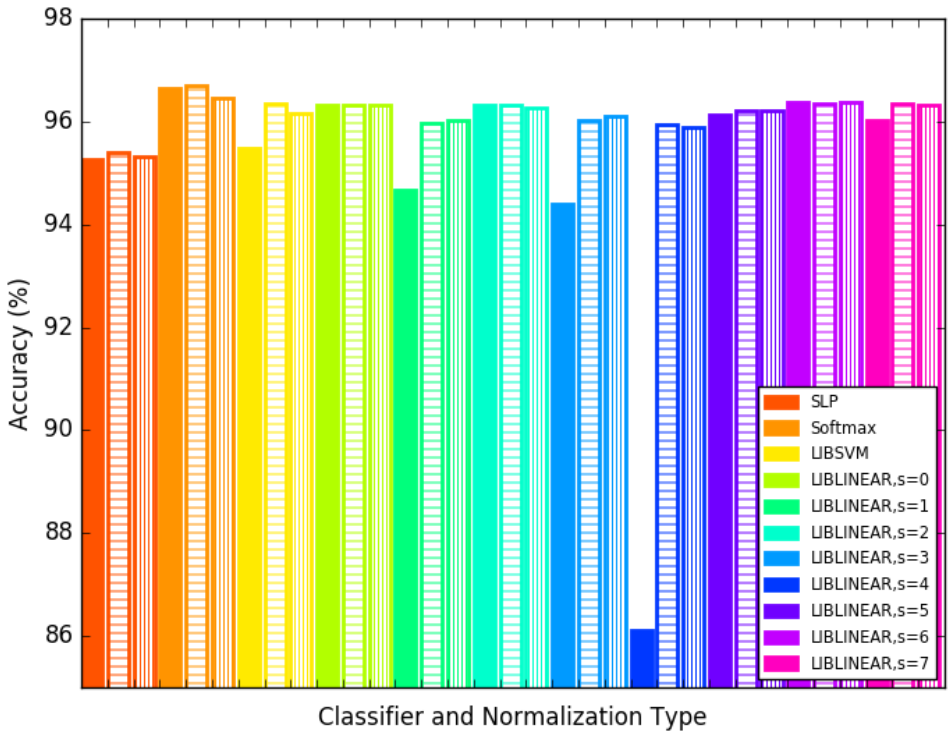

Figure 5.6: Mean classifier accuracy for the HFirst architecture on the N-MNIST dataset. Solid bars indicate no scaling, bars with horizontal lines indicate pattern-dimension scaling, and bars with vertical lines indicate feature-dimension scaling. In this task, both scaling techniques produce higher recognition accuracy than non-scaled inputs for most of the classifiers.

classifiers across datasets, so mean classification will have to suffice.

Clearly illustrated in Figure 5.8 is the superiority of the softmax classifier to the other linear classifiers, even when the inputs are not scaled. However, it is still advantageous to scale the inputs to the softmax classifier from an accuracy standpoint. The mean of the softmax results in Figure 5.8 across normalization types is 55.88%, which is 0.73% higher than the next-best performing classifier, LIBSVM's L2-regularized, L1-loss, "one-vs-one" linear classifier. This "one-vs-one" classifier outperformed its "one-vs-all" counterpart (LIBLINEAR, $s = 3$) by an average of

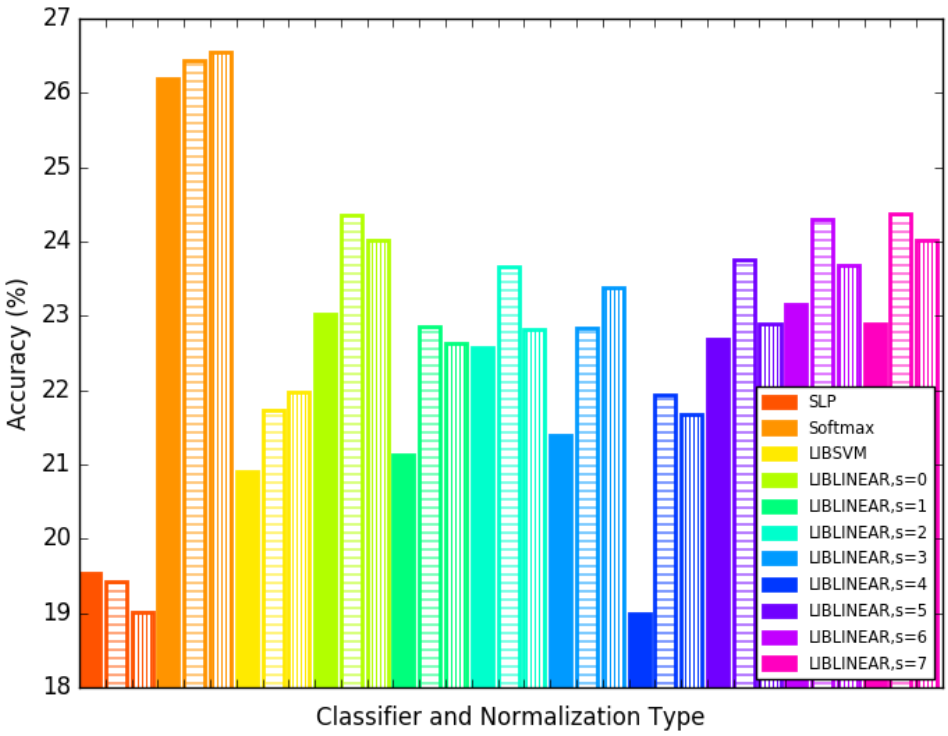

Figure 5.7: Mean classifier accuracy for the HFirst architecture on the CIFAR10-DVS dataset. Solid bars indicate no scaling, bars with horizontal lines indicate pattern-dimension scaling, and bars with vertical lines indicate feature-dimension scaling. The softmax classifier is clearly superior to the others in this case. Also, pattern-dimension scaling of the inputs is preferable to no scaling and feature-dimension scaling for most of the SVMs.

1.49% across normalization types. The linear classifiers from the LIBLINEAR package varied quite a bit in accuracy, and more work needs to be done to determine why the non-scaled input to Crammer and Singer's support vector classifier [2] ($s = 4$) did not perform well. On average (across normalizations), the softmax classifier outperformed the SLP by 5.28%. It becomes evident that the simplicity of the SLP's activation and cost functions corresponds to a weaker linear classifier.

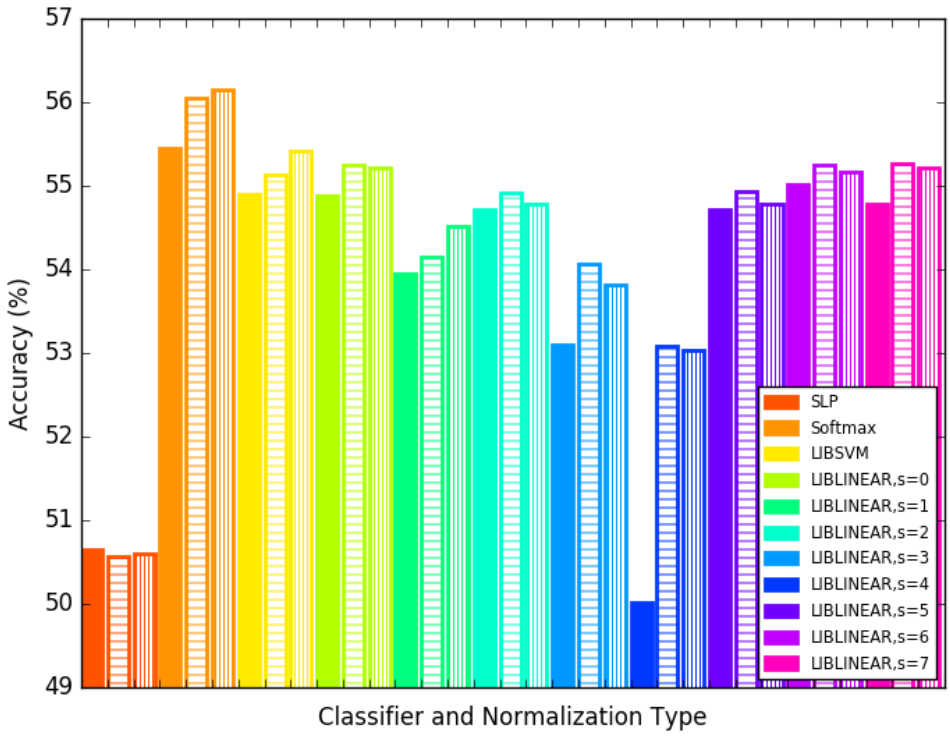

Figure 5.8: Mean classifier accuracy for both spiking models and both datasets. Solid bars indicate no scaling, bars with horizontal lines indicate pattern-dimension scaling, and bars with vertical lines indicate feature-dimension scaling. Feature-dimension min-max scaling of the softmax inputs results in the highest mean classification performance across both datasets and SNN architectures.

It is unclear whether feature-dimension scaling increases accuracy over pattern-dimension scaling in these classification tasks. Looking at Figure 5.8, either of these normalization techniques offers superior classification performance when compared to the case of no data normalization for all classifiers except for the SLP. However, there is no evidence that the feature-dimension min-max scaling as outlined in [7] for linear SVMs aided classification accuracy in these tasks.

5.5 Classifier Convergence

With stochastic gradient descent of softmax classifiers, it is important to train the network long enough so that it converges on a near-optimal solution. In order to verify that we are reaching this point, we look at testing accuracy at every training epoch for a typical experiment. Figure 5.9 shows this testing accuracy during all 1,500 epochs of training, and Figure 5.10 shows accuracy for the last 750 training epochs. The choice of learning rate (0.1) and number of training epochs (1,500) for the softmax classifier seemed to be a good choice of hyperparameters in order to learn a near-optimal set of weights.

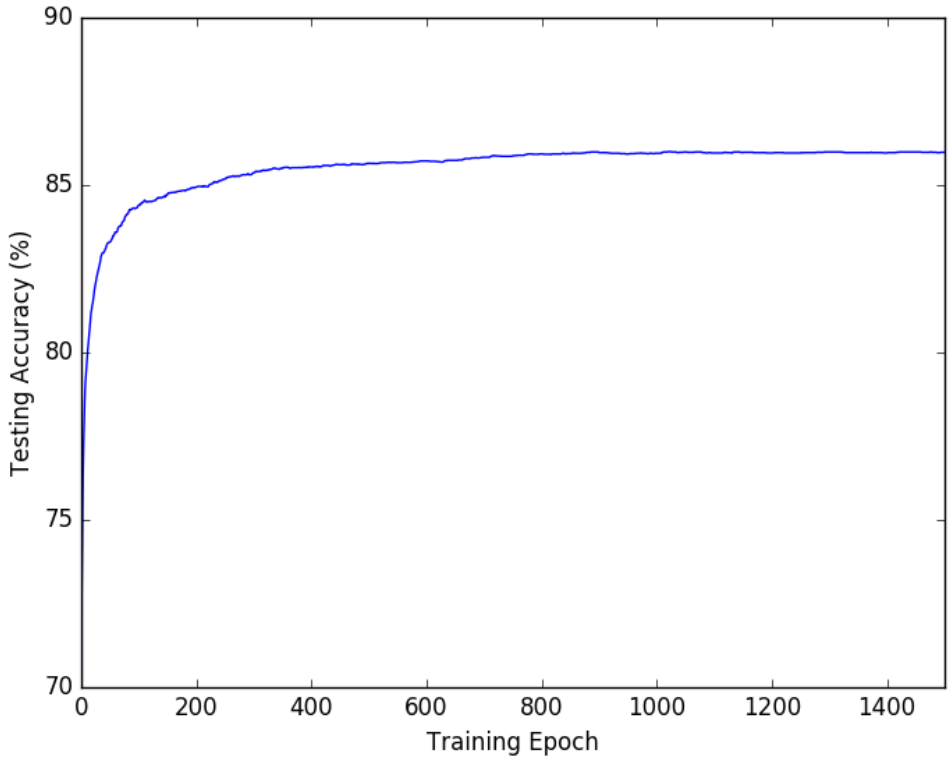

Figure 5.9: Testing accuracy at every epoch of softmax training. This example is learning SSLCA activations in response to the N-MNIST dataset.

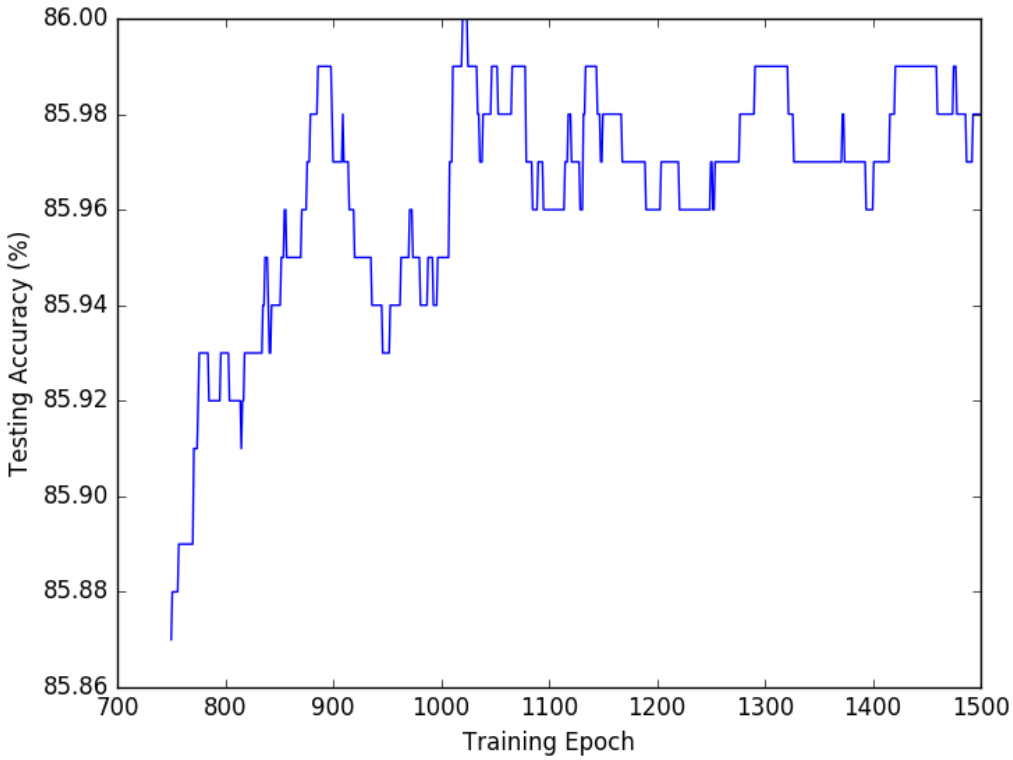

Figure 5.10: Testing accuracy at the last 750 epochs of softmax training. This example is learning SSLCA activations in response to the N-MNIST dataset.

In order to determine if our accuracy for all classifiers increases if training time is lengthened, we extended the classifier training time for all hyperparameter variations of SSLCA (126 experiments) on the N-MNIST dataset with feature-dimension scaling. We did not do this with the SLP classifier because its accuracy does not benefit from more training epochs. For the softmax classifier, we increased the number of training epochs from 1,500 to 10,000. For the LIBSVM and LIBLINEAR classifiers, we decrease the tolerance for a stopping condition ϵ, described in [6] for the LIBSVM classifier and [5] for the LIBLINEAR classifiers. For all the previous classifier experiments, we use their default values, shown in Table 5.7. In order to determine if "early stopping" played a factor in the mean classification

performance of the LIBSVM and LIBLINEAR classifiers, we re-trained these with an ϵ of 0.1 times the default ϵ of each classifier. The mean classification accuracy on the N-MNIST testing set for the softmax, LIBSVM, and LIBLINEAR classifiers with the previous and new stopping conditions is shown in Table 5.8. This seems to validate the results in Figure 5.8, as 10,000 training epochs or an ϵ value that is one tenth that of the earlier experiments does not increase classification performance substantially in any of the classifiers.

Classifier	Default ϵ
LIBSVM	0.1
LIBLINEAR, s=0	0.01
LIBLINEAR, s=1	0.1
LIBLINEAR, s=2	0.01
LIBLINEAR, s=3	0.1
LIBLINEAR, s=4	0.1
LIBLINEAR, s=5	0.01
LIBLINEAR, s=6	0.01
LIBLINEAR, s=7	0.1

Table 5.7: Default stopping tolerance ϵ for the SVM classifiers

Classifier	Mean Accuracy, Default Stopping	Mean Accuracy, Later Stopping
Softmax	82.61%	82.71%
LIBSVM	83.78%	83.78%
LIBLINEAR, s=0	81.28%	81.31%
LIBLINEAR, s=1	80.85%	80.85%
LIBLINEAR, s=2	80.90%	80.93%
LIBLINEAR, s=3	80.37%	80.35%
LIBLINEAR, s=4	82.18%	82.13%
LIBLINEAR, s=5	80.94%	80.95%
LIBLINEAR, s=6	81.39%	81.42%
LIBLINEAR, s=7	81.31%	81.31%

Table 5.8: Mean classifier accuracy for different stopping criteria for feature-scaled SSLCA activations in response to N-MNIST

Chapter 6

Conclusion

Spiking neural networks are well-suited to extract meaningful information from event-based data because, for one, they preserve the temporal dynamics of the input sequence. HFirst exploits these dynamics, while SSLCA doesn't explicitly. Also, information about the position of a basis function within a given input is preserved with HFirst, while SSLCA basis functions span the whole input in these experiments. Nevertheless, they are two exemplary low-power solutions to extracting important information in event-based data. To date, no comprehensive classifier comparisons have been performed for SNNs paired with event-based data. Given the sparse nature of the data, it is not apparent that the results of a linear classifier comparison that have been performed in response to static images would transfer to the event-based domain, as the sparsity of the neurons changes the "separability" of the input representation. Regardless, conflicting works exist in the domain of linear classifier performance. We show that, across datasets and SNN architectures, the softmax classifier seems to show modestly better performance to any method of linear SVM classifiers, who considerably outperform the SLP classifier. This last part makes sense, as SLPs only learn a solution, while SVMs learn a solution that maximizes the margin between the support vectors. Even in the case of non-separable data such as the spiking activity of HFirst and SSLCA, it learns a more robust solution that generalizes better to new data. More work needs to be done in order to determine why the softmax classifier seems to classify better. Perhaps it is due to the effect of mini-batching, the size of mini-batches,

the learning rate choice, the selection of scaling techniques, etc.

The SVMs used in this work were not optimized for their C parameter, which varies for different datasets. This is because we wish to compare classifiers with fixed hyperparameters (stopping condition, learning rate, number of examples in a mini-batch, etc.) in the case where optimizing every classifier hyperparameter would be too time consuming or computationally expensive, and we only want to choose between linear classifiers.

Non-linear classifiers perform at least as well as linear ones, but we did not explore them in this work, as we are interested in low-power systems. The addition of non-linear kernels significantly increases the classifier's number of computations, so it would not scale well.

We achieve state of the art classification on CIFAR10-DVS (29.67% to 31.76%) using the HFirst SNN (V_{thresh} = 100 mV/ms, refractory period = 1 ms, S1 decay rate = 20 mV/ms) paired with a linear SVM (LIBLINEAR, $s = 0$, $C = 1$, feature-scaled), and state of the art classification on N-MNIST (97.77% to 97.82%) with a convolutional SNN (HFirst, V_{thresh} = 100 mV/ms, refractory period = 1 ms, S1 decay rate = 20 mV/ms) paired with a linear SVM (LIBSVM, pattern-scaled, C = 1).

We improved HFirst on the N-MNIST dataset from 71.15% to 97.82% by replacing HFirst's linear classifier with a classifier that generalizes significantly better to new examples than one that performs template matching with normalized spike counts as the additive weights.

Of course, there are other stochastic gradient descent optimizations that could improve upon the accuracy reported here, such as cross validation, where parts of the training set are left out of the training phase and tested in order to prevent

overfitting of the training set, adaptive learning rates where the learning rate is not fixed during training, the use of a non-linear classifier, etc. These techniques are not explored in this work, but may have an effect on classification performance.

In all, we have provided a reasonable comparison of linear classifiers for spiking neural network rate-based activity in response to event-driven image data, we have performed an exploration of hyperparameters for two SNNs in order to determine sensitivity of classification performance to said hyperparameters, we have improved the HFirst architecture by substituting classifiers, and we have improved a state of the art event-driven softmax classifier. Our results are important for low-power SNNs, where the adoption of probability-based softmax classifier trained with mini-batches is imperative for the maximization of classification accuracy in event-based tasks.

References

[1] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[2] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(Dec):265–292, 2001.

[3] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch. Activity-driven, event-based vision sensors. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2426–2429. IEEE, 2010.

[4] C.B. Do and A.Y. Ng. Transfer learning for text classification. In *Advances in Neural Information Processing Systems*, pages 299–306, 2006.

[5] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[6] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. *Journal of machine learning research*, 6(Dec):1889–1918, 2005.

[7] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. 2003.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[9] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[10] J. Konečnỳ, J. Liu, P. Richtárik, and M. Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016.

[11] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] H. Li, H. Liu, X. Ji, G. Li, and L. Shi. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11, 2017.

[14] M. Li, T. Zhang, Y. Chen, and A.J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661–670. ACM, 2014.

[15] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 x 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[16] P.A. Merolla, J.V. Arthur, R. Alvarez-Icaza, A.S. Cassidy, J. Sawada, F. Akopyan, B.L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[17] E. Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

[18] B.A. Olshausen and D.J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997.

[19] G. Orchard, A. Jayawant, G.K. Cohen, and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, 2015.

[20] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman. Hfirst: a temporal approach to object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(10):2028–2040, 2015.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] C. Posch, D. Matolin, and R. Wohlgenannt. A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[23] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5(Jan):101–141, 2004.

[24] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[25] C.J. Rozell, D.H. Johnson, R.G. Baraniuk, and B.A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20(10):2526–2563, 2008.

[26] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in Neuroscience*, 11, 2017.

[27] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[28] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[29] W. Woods and C. Teuscher. Fast and accurate sparse coding of visual stimuli with a simple, ultra-low-energy spiking architecture. *arXiv preprint arXiv:1704.05877*, 2017.

[30] Y. Yamada, M. Iwamura, and K. Kise. Shakedrop regularization. *arXiv preprint arXiv:1802.02375*, 2018.

[31] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[32] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.