High Capacity Neural Block Classifiers with Logistic Neurons and Random Coding

Olaoluwa Adigun
Signal and Image Processing Institute
Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA 90089-2564.
adigun@usc.edu

Bart Kosko
Signal and Image Processing Institute
Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA 90089-2564.
kosko@usc.edu

***Abstract*—We show that neural networks with logistic output neurons and random codewords can store and classify far more patterns than those that use softmax neurons and 1-in-K encoding. Logistic neurons can choose binary codewords from an exponentially large set of codewords. Random coding picks the binary or bipolar codewords for training such deep classifier models. This method searched for the bipolar codewords that minimized the mean of an inter-codeword similarity measure. The method used *blocks* of networks with logistic input and output layers and with few hidden layers. Adding such blocks gave deeper networks and reduced the problem of vanishing gradients. It also improved learning because the input and output neurons of an interior block must equal the input pattern's code word. Deep-sweep training of the neural blocks further improved the classification accuracy. The networks trained on the CIFAR-100 and the Caltech-256 image datasets. Networks with 40 output logistic neurons and random coding achieved much of the accuracy of 100 softmax neurons on the CIFAR-100 patterns. Sufficiently deep random-coded networks with just 80 or more logistic output neurons had better accuracy on the Caltech-256 dataset than did deep networks with 256 softmax output neurons.**

***Index Terms*—logistic network, blocking, random coding, deep sweep training, backpropagation invariance**

I. Logistic versus Softmax Neurons for Large-Capacity Networks

We show that a network with logistic output neurons and random coding can store the same number K of patterns as a softmax classifier but with a smaller number M of output neurons. The logistic network's classification accuracy falls as M becomes much smaller than K. This implies that a properly coded logistic network can store far more patterns with similar accuracy than a softmax network can with the same number of outputs. We further show that randomly encoded logistic *blocks* lead to still more efficient deep networks.

Almost all deep classifiers map input patterns to K output softmax neurons. So they code the K pattern classes with K unit bit vectors and thus with 1-in-K coding. The softmax output layer has the likelihood structure of a one-shot multinomial probability or the single roll of K-sided die and thus its log-likelihood is the negative of the cross entropy [1], [2]. This softmax structure produces an output probability vector and so restricts its coding options to the K unit bit vectors of the K-dimensional unit hypercube $[0,1]^K$.

Logistic output coding can use any of the 2^K binary vertices of the hypercube $[0,1]^K$. This allows far fewer output logistic neurons to accurately code for the K pattern classes. The logistic layer's likelihood is that of a product of Bernoulli probabilities and thus K flips of a coin. Its log-likelihood has a double cross-entropy structure [1], [2]. The softmax and logistic networks coincide when $K = 1$.

Figure 1 shows the block structure of a deep logistic network. Figure 2 shows sample random coding vectors of lengths M = 20, 60, and 100 for logistic networks that encode $K = 100$ pattern classes. The remaining figures show how block logistic networks with random coding can encode the CIFAR-100 and Caltech-256 patterns with fewer than $K = 100$ or $K = 256$ respective output neurons. Logistic networks also had higher classification accuracy than did softmax networks with the same number of neurons. The last figure shows that the very best performance came from deep-sweep training of all the blocks after pre-training the individual blocks. Table 3 shows that 80 logistic output neurons did better on the Caltech-256 data than did 256 softmax output neurons.

Earlier work [3], [4] explored how random basis vectors affected the approximation error of neural function approximators. Our random coding method deals with increasing the capacity of encoding patterns at the output or $visible$ hidden layers. Other work [5] explored the formal capacity of some feedforward networks. Our work shows how to improve the pattern capacity of deep neural classifiers with logistic output neurons, block structure, and deep-sweep training.

II. Finding Random Codewords for Patterns

A. Network Likelihood Structure and BP Invariance

Training a neural network optimizes the network parameters with respect to an appropriate loss function. This also maximizes the log-likelihood $L(\mathbf{y}|\mathbf{x}, \mathbf{\Theta})$ of the network [6]–[8]. Backpropagation invariance holds at each layer if the parameter gradient of the layer likelihood gives back the same backpropagation learning laws [1], [2].

The network's complete likelihood desribes the joint probability of all layers [1]. Suppose a network has J hidden layers $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_J$. The term $\mathbf{h}_j$ denotes the j^{th} hidden layer after the input (identity) layer. The complete likelihood is

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

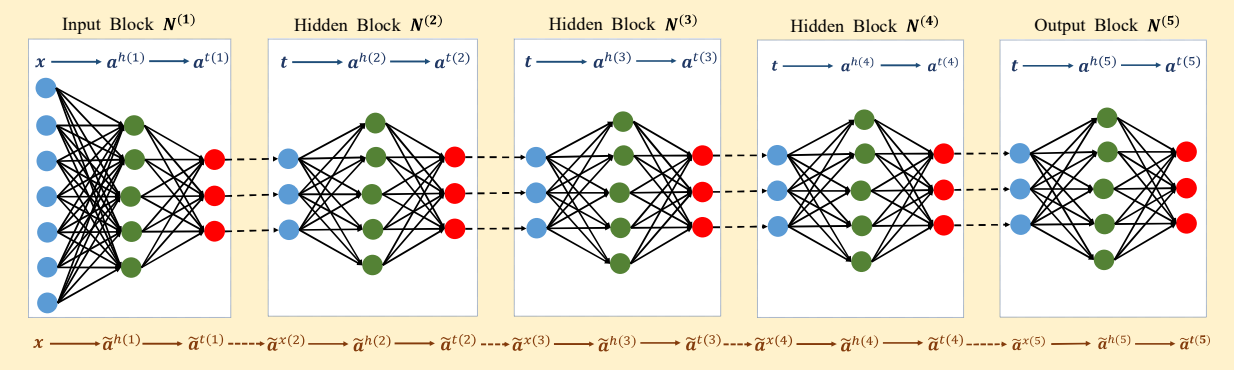


Fig. 1: Modular architecture of a deep block neural network. The deep-sweep training method in Algorithm 1 used blocking to break a deep neural network into small multiple blocks. The network had an input block $N^{(1)}$, three hidden blocks $\{N^{(2)}, N^{(3)}, N^{(4)}\}$, and output block $N^{(5)}$. Each block had three layers in the simplest case. The terms $a^{t(1)}, ..., a^{t(4)}$ represent the activations for the $visible$ hidden layers and $a^{t(5)}$ is the output activation. The terms $a^{h(1)}, ..., a^{h(5)}$ represent the activations of the $non\text{-}visible$ hidden layers. The deep-sweep method used two stages: pre-training and fine-tuning. The pre-training stage trained the blocks separately. It used supervised training for each block by using the block error $E^{(b)}$ between the output activation $\mathbf{a}^{t(b)}$ and the target $\mathbf{t}$. The fine-tuning stage began after the pre-training and also used supervised learning. It stacked all the blocks together and used an identity matrix I to connect contiguous blocks. Fine tuning optimized the weights with respect to the joint error E_{ds}.

the probability density $p(\mathbf{y}, \mathbf{h}_J, ..., \mathbf{h}_1 | \mathbf{x}, \Theta)$. The *chain rule* or *multiplication theorem* of probability factors the likelihood into a product of the layer likelihoods:

$$p(\mathbf{y}, \mathbf{h}_J, ..., \mathbf{h}_1 | \mathbf{x}, \Theta) = p(\mathbf{y} | \mathbf{h}_J, ..., \mathbf{h}_1, \mathbf{x}, \Theta) \times p(\mathbf{h}_1 | \mathbf{x}, \Theta) \times \prod_{j=2}^{J} p(\mathbf{h}_j | \mathbf{h}_{j-1}, ..., \mathbf{h}_1, \mathbf{x}, \Theta) \quad (1)$$

where we assume that $p(\mathbf{x}) = 1$ for simplicity [1], [9], [10]. So the complete log-likelihood $L(\mathbf{y}, \mathbf{h} | \Theta)$ is $L(\mathbf{y}, \mathbf{h} | \Theta) = \log p(\mathbf{y}, \mathbf{h}_J, ..., \mathbf{h}_1 | \mathbf{x}, \Theta) = L(\mathbf{y} | \mathbf{x}, \Theta) + \sum_{j=1}^{J} L(\mathbf{h}_j | \mathbf{x}, \Theta)$ where $L(\mathbf{h}_j | \mathbf{x}, \Theta) = \log p(\mathbf{h}_j | \mathbf{h}_{j-1}, ..., \mathbf{h}_1, \mathbf{x}, \Theta)$. The output layer has log-likelihood $L(\mathbf{y} | \mathbf{x}, \Theta) = \log p(\mathbf{y} | \mathbf{h}_J, ..., \mathbf{h}_1, \mathbf{x}, \Theta)$. The next sections use this structure in the equivalent form of layer error functions.

B. Output Activation, Decision Rule, and Error Function

Input $\mathbf{x}$ passes through a classifier network $\mathcal{N}$ and gives $\mathbf{o}^t = \mathcal{N}(\mathbf{a}^x)$ where $\mathbf{o}^t$ is the input to the output layer. The output activation $\mathbf{a}^t$ equals $f(\mathbf{o}^t)$ where f is a monotonic and differentiable function. Softmax or Gibbs activation functions [6], [11] remain the most used output activation for neural classifiers. This paper explores instead binary and bipolar output *logistic* activations. Logistic output activations give a choice of 2^M codewords at the vertices of the unit cube $[0, 1]^M$ to code for the K patterns as opposed to the softmax choice of just the M vertices of the embedded probability simplex.

Codeword $\mathbf{c}_k$ is an M-dimensional vector that represents the k^{th} class. M is the codeword length. Each target $\mathbf{t}$ is one of the K unique codewords $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_K\}$. The decision rule for classifying $\mathbf{x}$ maps the output activation $\mathbf{a}^t$ to the class with the closest codeword:

$$C(\mathbf{x}) = \arg\min_k \sum_{l=1}^{K} \left| c_{kl} - \ a_l^t \right| \quad (2)$$

where $C(\mathbf{x})$ is the predicted class for input $\mathbf{x}$, a_l^t is the l^{th} argument of the output activation, and c_{kl} is the l^{th} component of the k^{th} codeword $\mathbf{c}_k$. The next section describes the output activations and their layer-likelihood structure.

1) Softmax or Gibbs Activation: This activation maps the neuron's input $\mathbf{o}^t$ to a probability distribution over the predicted output classes [2], [11]. The activation a_l^t of the l^{th} output neuron has the mutli-class Bayesian form:

$$a_l^t = \frac{\exp(o_l^t)}{\sum_{k=1}^{K} \exp(o_k^t)} \quad (3)$$

where o_l^t is the input of the l^{th} output neuron. A single such logistic function defines the Bayesian posterior in terms of the log-posterior odds for simple two-class classification [6].

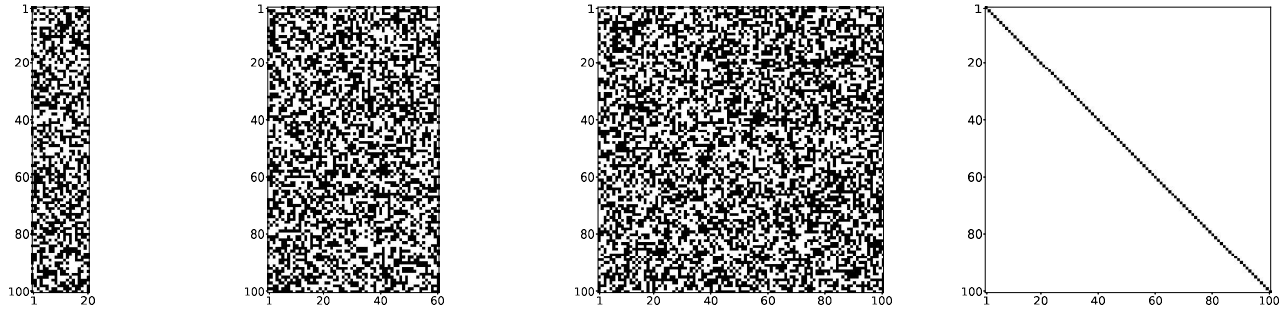
The softmax activation (3) uses K binary basis vectors from the Boolean $\{0, 1\}^K$ as the codewords. The codeword length M equals the number K of classes in this case: $M = K$. The decision rule follows from (2). The error function E_s for the softmax layer is the cross entropy [1] since it equals the negative of the log-likelihood for a layer multinomial likelihood—a single roll of the network's implied K-sided die:

$$E_s = -\sum_{k=1}^{K} t_k \log(a_k^t) = -\log \prod_{k=1}^{K} {a_k^t}^{t_k} \quad (4)$$

where t_k is the k^{th} argument of the target. The softmax decision rule follows from (2). The rule simplifies for the unit bit basis vectors as the codewords. Let $\sum_{l=1}^{K} \left| c_{kl} - a_l^t \right| = D^{(k)}$ where $D^{(k)}$ is the distance between $\mathbf{a}^t$ and $\mathbf{c}_k$. Then

$$C(\mathbf{x}) = \arg\min_k \ D^{(k)} = \arg\max_k \ a_k^t \quad (5)$$

because $M = K$. So $C(\mathbf{x}) = m$ implies that $D^{(m)} \leq D^{(k)}$ for $k \in \{1, 2, ..., K\}$. The decision rule simplifies as in (5) because $c_{kk} = 1$, $c_{kl} = 0$ for all $l \neq k$, and $0 \leq a_l^t \leq 1$ for $l \in \{1, 2, ..., K\}$.



(a) $M = 20$ (b) $M = 60$ (c) M = 100 (d) Basis vectors

Fig. 2: Bipolar codewords generated from the random coding method in Algorithm 1 with $p = 0.5$, $M \leq 100$, and $K = 100$. The algorithm found the set of codewords $\mathbf{C}^*$ with the smallest mean μ_c of the inter-codeword similarity measure d_{kl} . We searched for the best such random code words in 10,000 iterations. This figure shows the grayscale image of some of the codewords. The black pixels denote the bit value 1 and the white pixels denote the bit value -1. (a) shows the best code $\mathbf{C}^*$ with $M = 20$. (b) shows the best code $\mathbf{C}^*$ with $M = 60$. (c) shows the best code $\mathbf{C}^*$ with $M = 100$. (d) shows the 100 equidistant unit basis-vector codewords from the bipolar Boolean cube $\{-1, 1\}^{100}$ with $M = 100$.

2) Binary Logistic Activation: The binary activation a_l^t maps the input $\mathbf{o}^t$ to a vector in the unit hypercube $[0, 1]^M$:

$$a_l^t = \frac{1}{1 + \exp(-o_l^t)} \tag{6}$$

activation of the l^{th} output neuron where o_l^t is the input of the l^{th} output neuron. The codewords are vectors from $\{0, 1\}^M$ where $\log_2 K \leq M$. The decision rule for the bipolar logistic activation follows from (2). We can also impose the equidistant condition on the codewords by picking the basis vectors from the Boolean $\{0, 1\}^M$ as the codewords with $M = K$. The decision rule simplifies to equation (5) in this case. Binary logistic activation uses the double cross entropy E_{log} as its error function. This is equivalent to the negative of the log-likelihood with independent Bernoulli probability distribution.

$$E_{log} = -\sum_{l=1}^{M} \left[t_l \log(a_l^t) + (1 - t_l) \log(1 - a_l^t) \right] \tag{7}$$

$$= -\log \prod_{l=1}^{M} {a_k^t}^{(t_k)} 1 - {a_k^t}^{(1-t_k)} \tag{8}$$

The term a_l^t denotes the activation of the l^{th} output neuron and t_l is the l^{th} argument of the target vector.

3) Bipolar Logistic Activations: A bipolar logistic activation maps $\mathbf{o}^t$ to a vector in $[-1, 1]^M$. The activation a_l^t of the l^{th} output neuron has the form

$$a_l^t = \frac{2}{1 + \exp(-o_l^t)} - 1 = \frac{1 - \exp(-o_l^t)}{1 + \exp(-o_l^t)} \tag{9}$$

where o_l^t is the input into the l^{th} output neuron. The codewords are K bipolar vectors from $\{-1, 1\}^M$ such that $\log_2 K \leq M$.

The decision in this case follows from (2). The corresponding error function E_{b_log} is the double cross entropy. This requires a linear transformation of a_k^t and t_k as follows: $\tilde{a}_k^t = \frac{1}{2}(1 + a_k^t)$ and $\tilde{t}_k = \frac{1}{2}(1 + t_k)$. The bipolar logistic activation uses the transformed double cross-entropy. This is equivalent to the negative of the log-likelihood of the transformed terms with independent Bernoulli probabilities:

$$E_t = -\sum_{l=1}^{M} \left[\tilde{t}_k \log(\tilde{a}_k^t) + (1 - \tilde{t}_k) \log(1 - \tilde{a}_k^t) \right] \tag{10}$$

$$= -\frac{1}{2} \sum_{k=1}^{M} \Big[(1 + t_k) \log(1 + a_k^t) + (1 - t_k) \log(1 - a_k^t) - 2 \log 2) \Big] \tag{11}$$

$$= -\log \prod_{l=1}^{M} (\tilde{a}_k^t)^{(\tilde{t}_k)} (1 - \tilde{a}_k^t)^{(1-\tilde{t}_k)}. \tag{12}$$

Training seeks the best parameter Θ^* that minimizes the error function. So we can drop the constant terms in E_t. The modified error E_{b_log} has the form

$$E_{b\_log} = -\sum_{l=1}^{M} (1 + t_l) \log(1 + a_l^t) + (1 - t_l) \log(1 - a_l^t). \tag{13}$$

The backpropagation (BP) learning laws remain invariant at a softmax or logistic layer if the error functions have the appropriate respecitive cross-entropy or double-cross-entropy form. The learning laws are invariant for softmax and binary logistic activations because [7], [8]:

$$\frac{\partial E_s}{\partial u_{lj}} = \frac{\partial E_{log}}{\partial u_{lj}} = (a_l^t - t_l) a_j^h \tag{14}$$

where u_{lj} is the weight connecting the j^{th} neuron of the hidden layer to the l^{th} output neuron, a_j^h is the activation of the j^{th} neuron of the hidden layer linked to the output layer,

and $o_l^t = \sum_{j=1}^{J} u_{lj} a_j^h$. The derivative in the case of using a bipolar logistic output activation is

$$\frac{\partial E_{b\_log}}{\partial u_{lj}} = \frac{\partial E_{b\_log}}{\partial a_l^t} \frac{\partial a_l^t}{\partial o_l^t} \frac{\partial o_l^t}{\partial u_{lj}} \tag{15}$$

$$= \frac{2(a_l^t - t_k)}{(1 - a_l^t)(1 + a_l^t)} \frac{(1 + a_l^t)(1 - a_l^t)}{2} a_j^h \tag{16}$$

$$= (a_l^t - t_l) a_j^h. \tag{17}$$

So the BP learning laws remain invariant for the softmax, binary logistic, and bipolar logistic activations because (14) equals (17).

C. Random Coding with Bipolar Codewords

We now present the method for picking K random bipolar codewords from $\{-1, 1\}^M$ with $\log_2 K \leq M < K$. The bipolar Boolean cube contains 2^M codewords since the bipolar unit cube $[-1, 1]^M$ has M vertices. It is computationally expensive to pick $M = K$ for a dataset with big values of K such as 10,000 or more [12], [13]. Our goal is to find an efficient way to pick K codewords with $\log_2 K \leq M < K$.

Let code $\mathbf{C}$ be a $K \times M$ matrix such that the k^{th} row $\mathbf{c}_k$ is the k^{th} codeword and d_{kl} be the similarity measure between $\mathbf{c}_k$ and $\mathbf{c}_l$. We have $d_{kl} = \left|\mathbf{c}_k \cdot \mathbf{c}_l\right|$. There are $\frac{1}{2}(K(K-1))$ unique pairs of codewords. The mean μ_c of the inter-codeword similarity measure has the normalized correlation form

$$\mu_c = \frac{2}{K(K-1)} \sum_{k=1}^{K} \sum_{l>k}^{K} \left|\mathbf{c}_k \cdot \mathbf{c}_l\right| . \tag{18}$$

This random coding method uses μ_c to guide the search. The method finds the best code $\mathbf{C}^*$ with the minimum similarity mean μ_c^* for a fixed M. Algorithm 1 shows the pseudocode for this method. A high value of μ_c implies that most of the codewords are not orthogonal while a low value of μ_c implies that most of the codewords are orthogonal. Figure 2 shows examples of codewords from Algorithm 1.

D. Deep-Sweep Training of Blocks

Deep-sweep training optimizes a network with respect to the network's complete likelihood in (1). This method performs *blocking* on deep networks by breaking the network down into small multiple contiguous networks or blocks. Figure 1 shows the architecture of a deep neural network with the deep-sweep training method. The figure shows the small blocks that make up the deep neural network. $N^{(1)}$ is the input block, $N^{(B)}$ is the output block, and the others are hidden blocks. The layer of connection between two blocks is treated as a *visible* hidden layer. We need the number of blocks $B \geq 2$ to use the deep-sweep method. Let the term $L^{(b)}$ denote the number of layers for block $N^{(b)}$. $L^{(b)}$ must be greater than 1 because each block has at least an input layer and an output layer. Θ_b represents the weights of $N^{(b)}$.

The deep-sweep training method trains a neural network in two stages. The first stage is the pre-training and the second stage is fine-tuning. The pre-training stage trains the blocks separately as supervised learning tasks. $N^{(1)}$ maps $\mathbf{x}$ into

Algorithm 1 : Random coding search w.r.t. the mean μ_c of the similarity measure with bipolar codewords.

Require: Code length M, number of codewords K, number of search iterations T, and sampling probability p.

$\mu_c{}^*$ = 0.0
for $t = 1$ to T **do**
  Pick K distinct codewords of length M:
  for $k = 1$ to K **do**
    Randomly generate a codeword from $\{-1, 1\}^M$:
    if k == 1 **then**
      Generate a $(1 \times M)$ vector $\mathbf{C}^{(t)}$ by picking M samples with replacement from $\{-1, 1\}$ with the probability p.
    else
      $search\_status$ = True
      while $search\_status$ == True **do**
        Generate a $(1 \times M)$ vector $\mathbf{c}_k$ by picking M samples with replacement from $\{-1, 1\}$ with the probability p.
        Compute the inter-codeword similarity measures between $\mathbf{c}_k$ and $\mathbf{C}^{(t-1)}$:

$$\boldsymbol{d}_k = \mathbf{C}^{(t-1)} \ \mathbf{c}_k{}^T.$$

        Drop $\mathbf{c}_k$ if there is a copy in $\mathbf{C}^{(t-1)}$.
        if $\max(\boldsymbol{d}_k) < M$ **then**
          $\mathbf{C}^{(t)} \Leftarrow$ Stack $\mathbf{c}_k$ and $\mathbf{C}^{(t-1)}$ to form a $(k \times M)$ matrix.
          $search\_status$ = False
        end if
      end while
    end if
  end for
  Compute the mean $\mu_c^{(t)}$ for code $\mathbf{C}^{(t)}$ using (18).
  if $(t == 1)$ or $(\mu_c^* < \mu_c^{(t)})$ **then**
    Update the best code $\mathbf{C}^*$ and best mean $\mu_c{}^*$.
  end if
end for

the corresponding range of the output activation. The output activation $\mathbf{a}^{t(b)}$ of the b^{th} block is:

$$\mathbf{o}^{t(b)} = \begin{cases} N^{(b)}(\mathbf{t}), & \text{if } b \in \{2, ..., B\} \\ N^{(b)}(\mathbf{x}), & \text{otherwise} \end{cases} \tag{19}$$

and $\mathbf{a}^{t(b)} = f\left(\mathbf{o}^{t(b)}\right)$ where $\mathbf{t}$ is the target, $\mathbf{o}^{t(b)}$ is the input to the output layer of $N^{(b)}$, and $\mathbf{a}^{t(b)}$ is the output activation of $N^{(b)}$. The error function $E^{(b)}$ measures the error between the target $\mathbf{t}$ and activation $\mathbf{a}^{t(b)}$. The error function $E^{(b)}$ of $N^{(b)}$ for $b \in \{1, 2, 3, .., B\}$ with a bipolar logistic activation is:

$$E^{(b)} = -\sum_{l=1}^{M} (1 + t_l) \log\left(1 + a_l^{t(b)}\right) + (1 - t_l) \log\left(1 - a_l^{t(b)}\right) \tag{20}$$

where $a_l^{t(b)}$ is the l^{th} component of the output activation of $N^{(b)}$. The fine-tuning stage follows the pre-training stage. It involves stacking the blocks and a deep-sweep across the entire network $\mathcal{N}$ from the input layer to the ouptut layer. Figure 1 shows the stacked blocks where $\mathbf{x}$ is the input through $N^{(1)}$ and the output activation $\tilde{\mathbf{a}}^{t(B)}$ comes from the output of $\mathcal{N}$. We have:

$$\tilde{\mathbf{o}}^{t(b)} = \begin{cases} N^{(b)}\big(\tilde{\mathbf{a}}^{t(b-1)}\big), & \text{if } b \in \{2, ..., B\} \\ N^{(b)}(\mathbf{x}), & \text{otherwise} \end{cases} \tag{21}$$

and $\tilde{\mathbf{a}}^{t(b)} = f\big(\tilde{\mathbf{o}}^{t(b)}\big)$. The deep-sweep error $E_{ds}^{(b)}$ for the fine-tuning stage is different from the error $E^{(b)}$. $E_{ds}^{(b)}$ is the deep-sweep error between $\tilde{\mathbf{a}}^{t(b)}$ and the target $\mathbf{t}$. So the corresponding deep-sweep error for a network with bipolar logistic activation is:

$$E_{ds}^{(b)} = -\sum_{l=1}^{M} \Big[(1 + t_l) \log\big(1 + \tilde{a}_l^{t(b)}\big) + (1 - t_l) \log\big(1 - \tilde{a}_l^{t(b)}\big)\Big] \tag{22}$$

for $b \in \{1, 2, ..., B\}$ where $\tilde{a}_l^{t(b)}$ is the l^{th} component of the activation $\tilde{\mathbf{a}}^{t(b)}$. The update rule at this stage differs from ordinary BP. Ordinary BP trains network parameters with a single error function at the output layer since the algorithm does not directly *know* the correct output value of a hidden layer. But we do know the correct output layer of an interior block since it just equals the random codeword. So the deep-sweep method updates the weights with respect to errors at the output layer of the blocks. The joint deep-sweep error E_{ds} is:

$$E_{ds} = -\sum_{b=1}^{B}\sum_{l=1}^{M} \Big[(1 + t_l) \log\big(1 + \tilde{a}_l^{t(b)}\big) + (1 - t_l) \log\big(1 - \tilde{a}_l^{t(b)}\big)\Big] \tag{23}$$

$$= \sum_{b=1}^{B} E_{ds}^{(b)} \tag{24}$$

and the update rule for any parameter Θ_b follows from the derivative of this joint error. Algorithm 2 shows the pseudocode for this method.

III. Simulation Experiments

Our coding simulations compared the performance of the output activations. Output logistic activations outperformed softmax activation. We also simulated the performance of the random coding method in algorithm 1. The classification accuracy of neural classifiers decreased as μ_c increased with a fixed M and $\log_2 \leq M < K$. The result also shows that the accuracy with bipolar codewords and $M = 0.4K$ is comparable with the accuracy from using the softmax activation with K-dimensional codewords (basis vectors).

We found that training a deep neural classifier with the deep-sweep method outperformed training with ordinary back-propagation. The next sections describes the datasets for the experiments.

Algorithm 2 : Deep-sweep training algorithm.

Require: Learning rate η, batch size M, training epochs N, iterations per epoch R, number of blocks B, size of blocks $\{L^{(1)}, ..., L^{(B)}\}$, and start of fine-tuning stage n_0.

Require: Initial weights for the blocks $\Theta_1^{(0)}$, $\Theta_2^{(0)}$, ... , $\Theta_B^{(0)}$ and the training samples $\{\mathbf{x}_j,\ \mathbf{t}_j\}_{j=1}^{J}$.

$num\_of\_iters = N \times R$
for $r = 1$ to num_of_iters **do**
  Select a batch of M samples $\{\mathbf{x}_m,\ \mathbf{t}_m\}_{m=1}^{M}$.
  if $r \leq n_0$ **then**
    for $b = 1$ to B **do**
      Compute the output activatiom $\mathbf{a}_m^{t(b)}$ for input $\mathbf{x}_m$.
      Compute the pre-training error $E^{(b)}$:

$$E^{(b)} = -\frac{1}{M}\sum_{m=1}^{M} \Big[(\mathbf{1} + \mathbf{t}_m)^T \log\big(\mathbf{1} + \mathbf{a}_m^{t(b)}\big) + (\mathbf{1} - \mathbf{t}_m)^T \log\big(\mathbf{1} - \mathbf{a}_m^{t(b)}\big)\Big].$$

      Update the block parameter Θ_b :

$$\Theta_b^{(r+1)} = \Theta_b^{(r)} - \eta \nabla_{\Theta_b} E^{(b)}\Big|_{\Theta_b = \Theta_b^{(r)}}.$$

    end for
  else
    Stack the B blocks into a single deep network.
    for $b = 1$ to B **do**
      Compute the output activation $\tilde{\mathbf{a}}_m^{t(b)}$ for input $\mathbf{x}_m$.
      Compute the deep-sweep error $E_{ds}^{(b)}$:

$$E_{ds}^{(b)} = -\frac{1}{M}\sum_{m=1}^{M} \Big[(\mathbf{1} + \mathbf{t}_m)^T \log\big(\mathbf{1} + \tilde{\mathbf{a}}_m^{t(b)}\big) + (\mathbf{1} - \mathbf{t}_m)^T \log\big(\mathbf{1} - \tilde{\mathbf{a}}_m^{t(b)}\big)\Big].$$

    end for
    Compute the joint deep-sweep error E_{ds} using equation (24).
    Update the all the weights as follows:

$$\Theta^{(r+1)} = \Theta^{(r)} - \eta \nabla_{\Theta} E_{ds}\Big|_{\Theta = \Theta^{(r)}}.$$

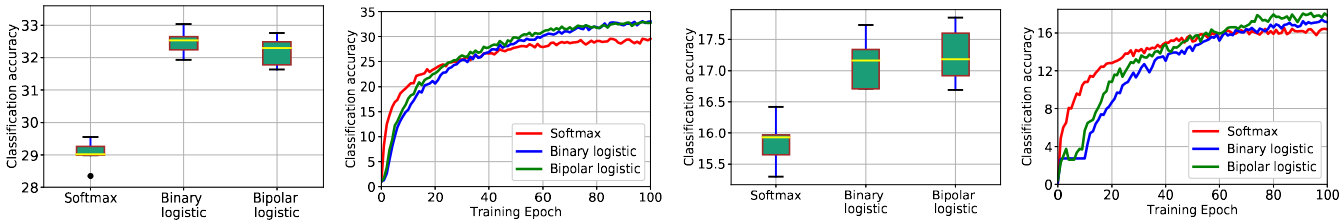
  end if
end for

A. Datasets

This classification experiments used the CIFAR-100 and Caltech-256 image datasets.

1) CIFAR-100: CIFAR-100 is a set of 60,000 color images from 100 pattern classes with 600 images per class. The 100 classes divide into 20 superclasses. Each superclass consists of 5 classes [14]. Each image has dimension 32×32×3. We used a 6-fold validation split with this dataset.

2) Caltech-256: This dataset had 30,607 images from 256 pattern classes. Each class had between 31 and 80 images. The 256 classes consisted of the two superclasses *animate* and *inanimate*. The animate superclass contained 69 pattern



(a) CIFAR-100 (b) CIFAR-100 (c) Caltech-256 (d) Caltech-256

Fig. 3: Logistic activations outperformed softmax activations for the same number K of output neurons. We compared the classifier accuracy of networks that used output softmax, binary logistic, and bipolar logistic neurons. Pattern coding used K binary basis vectors from the Boolean $\{0,1\}^K$ as the codewords for softmax or binary logistic outputs. Coding used K bipolar basis vectors from the bipolar cube $\{-1,1\}^K$ as the codewords for bipolar logistic outputs. Ordinary unidirectional backpropagation trained the networks. (a) shows the classification accuracy of the neural classifiers trained on the CIFAR-100 dataset with $K = 100$ where each model used 5-hidden layers with 512 neurons each. (b) shows the performance of the best model for each activation type. (c) shows the classification accuracy of the neural classifiers trained on the Caltech-256 dataset with $K = 256$ where each model used 7-hidden layers with 1,024 neurons each. (d) shows the performance of the best model (for each activation) with 7 hidden layers.

TABLE I: Output logistic activations outperformed softmax activations for the same number of output neurons. We used K binary basis vectors from the Boolean $\{0,1\}^K$ as the codewords with softmax or binary logistic activations. We used K bipolar basis vectors from the bipolar cube $\{-1,1\}^K$ as the codewords for bipolar logistic outputs. Ordinary backpropagation trained the classifiers. $K = 100$ for the CIFAR-100 dataset and $K = 256$ for the Caltech-256 dataset.

No. of Hidden Layers	**CIFAR-100**			**Caltech-256**		
	Softmax	***Binary Logistic***	***Bipolar Logistic***	***Softmax***	***Binary Logistic***	***Bipolar Logistic***
3 layers	$30.38 \pm 1.42\%$	$\mathbf{32.92 \pm 0.44\%}$	$32.65 \pm 0.83\%$	$17.01 \pm 0.92\%$	$\mathbf{19.43 \pm 0.73\%}$	$19.06 \pm 1.00\%$
5 layers	$29.04 \pm 0.74\%$	$\mathbf{32.47 \pm 0.73\%}$	$32.19 \pm 0.85\%$	$15.82 \pm 0.74\%$	$\mathbf{18.59 \pm 0.93\%}$	$17.93 \pm 0.70\%$
7 layers	$27.80 \pm 0.74\%$	$29.64 \pm 1.23\%$	$\mathbf{29.89 \pm 0.78\%}$	$15.19 \pm 0.70\%$	$\mathbf{18.08 \pm 0.81\%}$	$17.61 \pm 0.91\%$
9 layers	$26.58 \pm 0.61\%$	$26.77 \pm 1.12\%$	$\mathbf{27.47 \pm 0.68\%}$	$15.84 \pm 0.74\%$	$17.16 \pm 0.78\%$	$\mathbf{17.25 \pm 0.85\%}$

TABLE II: Random bipolar coding scheme with neural classifiers. The classifiers trained with random bipolar codewords from Algorithm 1 and used 5 hidden layers per model. We used code length $M = 30$ with the CIFAR-100 dataset and code length $M = 80$ with the Caltech-256 dataset. We used probability p to pick M samples with replacement from $\{-1,1\}$ when choosing the codewords. The mean μ_c of the similarity measure decreased as p increased from 0 to 0.5. The classification accuracy increased as the value μ_c decreased for a fixed value of M.

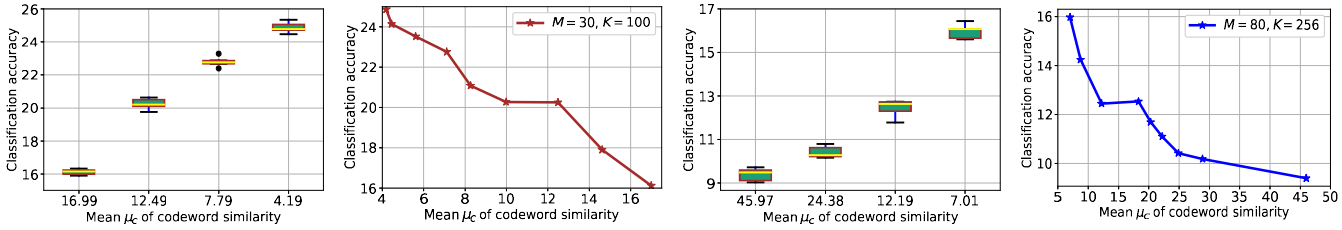
Dataset	**Probability** p	**Mean** μ_c	**Accuracy**
CIFAR-100	0.1000	16.9915	$16.12 \pm 0.31\%$
	0.1250	14.6145	$17.90 \pm 0.43\%$
	0.1500	12.4865	$20.25 \pm 0.60\%$
	0.1875	9.9794	$20.27 \pm 0.47\%$
	0.2125	8.2638	$21.08 \pm 0.46\%$
	0.2250	7.7931	$22.80 \pm 0.54\%$
	0.2375	7.1180	$22.75 \pm 0.76\%$
	0.2750	5.6162	$23.52 \pm 1.03\%$
	0.3500	4.4537	$24.13 \pm 0.61\%$
	0.5000	**4.1923**	$\mathbf{24.86 \pm 0.59\%}$
Caltech-256	0.1125	45.969	$9.392 \pm 0.54\%$
	0.1875	28.867	$10.18 \pm 1.04\%$
	0.2125	24.382	$10.42 \pm 0.50\%$
	0.2250	22.205	$11.10 \pm 1.08\%$
	0.2375	20.292	$11.69 \pm 0.54\%$
	0.2500	18.293	$12.53 \pm 0.86\%$
	0.3000	12.189	$12.44 \pm 0.72\%$
	0.3500	8.7057	$14.24 \pm 0.51\%$
	0.5000	**7.0074**	$\mathbf{15.97 \pm 0.62\%}$

classes. The inanimate superclass contained 187 pattern classes [15]. We removed the *cluttered* images and reduced the size of the dataset to 29,780 images. We resized each image to $100 \times 100 \times 3$. We used a 5-fold validation split with this case.

B. Network Description

We trained several deep neural classifiers on the CIFAR-100 and Caltech-256 datasets. The classifiers used 3,072 input neurons and $K = 100$ if they trained on the CIFAR-100 data. All the classifiers we trained on the CIFAR-100 had 512 neurons per hidden layer. The hidden neurons used ReLU activations of the form $a(x) = \max(0, x)$ although logistic hidden units also performed well in blocks. We trained some classifiers with the ordinary BP [14], [16] and then further trained others with the deep-sweep method. We used dropout pruning method for the hidden layers [17]. A dropout value of 0.9 for the *non-visible* hidden layers reduced overfitting. We did not use a dropout with the *visible* hidden layers.

The neural classifiers differed when trained on the Caltech-256 dataset. We used 30,000 neurons at the input layer and K equals 256 of the deep classifiers trained on this dataset. All the models trained on Caltech-256 used 1,024 neurons per hidden layer with the ReLU activation. We varied the value of code length M for the models with the bipolar logistic activation such that $\log_2 256 \leq M \leq 256$. We trained some classifiers with the ordinary BP and others with the deep-sweep method. The deep neural classifiers used 30,000 input neurons and M output neurons. Dropout pruned all the *non-visible* hidden layers with a dropout value of 0.8. We did not use a dropout with the *visible* hidden layers.



(a) CIFAR-100 (b) CIFAR-100 (c) Caltech-256 (d) Caltech-256

Fig. 4: Random bipolar coding with neural classifiers. Classification accuracy fell with an increase in the mean μ_c of inter-codeword similarity measure for a fixed code length M. The trained neural classifiers used 5 hidden layers with 512 neurons each and had code length $M = 30$ on the CIFAR-100 dataset. The trained neural classifiers used 5 hidden layers with 1,024 neurons each and had code length $M = 80$ on the Caltech-256 dataset. The random coding method in Algorithm 1 picked the codewords. We compared the effect of μ_c on the classification accuracy. (a) shows the accuracy when training the classifiers with the codewords from Algorithm 1. (b) shows that the accuracy decreased with an increase in μ_c for a fixed code length $M = 30$. (c) shows the accuracy when training the classifiers with the codewords from Algorithm 1. (d) shows that the accuracy decreased with an increase in μ_c for a fixed code length $M = 80$.

TABLE III: Using the bipolar codewords with small codeword length and logistic outputs gave a classifier accuracy comparable to that of using softmax outputs and K binary basis vectors from $\{0, 1\}^K$. The deep neural classifiers trained with bipolar codewords from Algorithm 1 on the CIFAR-100 and Caltech-256 datasets. We compared the performance of these classifiers to the accuracy of the models trained with K-basis vectors and softmax activations (from Table I). Training models on the CIFAR-100 dataset with bipolar codewords of length $M = 40 = 0.4K$ achieved between $88\% - 90\%$ of the accuracy obtained from using 100 binary basis vectors and softmax outputs. Training models on Caltech-256 dataset with bipolar codewords of length $M = 80 = 0.3125K$ achieved between $84\% - 101\%$ of the accuracy obtained from using the 256 binary basis vectors and softmax outputs (from Table I). It outperformed softmax activations in some cases with the Caltech-256 dataset.

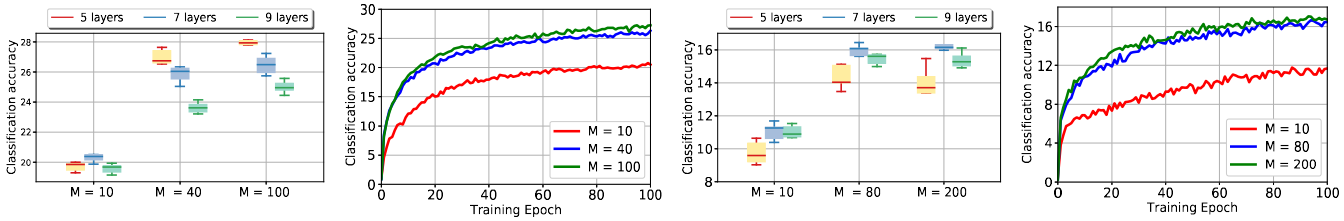
Dataset	**Code Length** M	**Best mean** μ_c^*	**Classification Accuracy**		
			3 Hidden Layers	***5 Hidden Layers***	***7 Hiddden Layers***
CIFAR-100	8	2.1156	$16.68 \pm 0.70\%$	$17.55 \pm 0.46\%$	$17.44 \pm 0.64\%$
	10	2.3733	$19.63 \pm 0.45\%$	$20.28 \pm 0.47\%$	$19.57 \pm 0.55\%$
	20	3.3774	$24.24 \pm 0.54\%$	$23.43 \pm 0.80\%$	$23.08 \pm 0.49\%$
	30	4.1923	$25.80 \pm 0.94\%$	$24.86 \pm 0.59\%$	$23.78 \pm 0.59\%$
	40	**4.8158**	$\mathbf{26.98 \pm 0.92\%}$	$\mathbf{25.86 \pm 0.95\%}$	$\mathbf{24.82 \pm 0.47\%}$
	50	5.3830	$27.61 \pm 0.82\%$	$26.33 \pm 0.60\%$	$25.03 \pm 0.54\%$
	60	5.9079	$27.63 \pm 0.74\%$	$26.52 \pm 0.84\%$	$25.57 \pm 0.63\%$
	70	6.3766	$27.78 \pm 0.64\%$	$26.82 \pm 0.71\%$	$25.23 \pm 0.80\%$
	80	6.8505	$27.62 \pm 0.99\%$	$26.41 \pm 0.79\%$	$25.14 \pm 1.08\%$
	90	7.2509	$27.86 \pm 0.82\%$	$26.84 \pm 0.52\%$	$25.26 \pm 0.56\%$
	100	7.6444	$27.80 \pm 0.84\%$	$26.47 \pm 1.11\%$	$25.01 \pm 0.73\%$
Caltech-256	10	2.4287	$9.78 \pm 1.25\%$	$11.05 \pm 0.95\%$	$11.03 \pm 0.67\%$
	20	3.4642	$12.56 \pm 1.49\%$	$13.78 \pm 1.02\%$	$13.70 \pm 0.66\%$
	50	5.5320	$13.82 \pm 1.17\%$	$15.63 \pm 1.05\%$	$14.99 \pm 0.81\%$
	80	**7.0074**	$\mathbf{14.28 \pm 1.27\%}$	$\mathbf{15.97 \pm 0.62\%}$	$\mathbf{15.43 \pm 0.59\%}$
	100	7.8234	$14.13 \pm 1.69\%$	$15.94 \pm 0.75\%$	$15.42 \pm 0.93\%$
	150	9.5934	$14.36 \pm 1.55\%$	$16.22 \pm 0.94\%$	$15.39 \pm 0.91\%$
	200	11.101	$14.36 \pm 1.69\%$	$16.25 \pm 0.55\%$	$16.06 \pm 0.58\%$
	250	12.401	$14.06 \pm 1.09\%$	$16.36 \pm 1.12\%$	$16.70 \pm 0.97\%$

C. Results and Discussion

Table I compares the effect of the output activations on the classification accuracy of deep neural classifiers. It shows that the logistic activations outperformed the softmax activation. We used the K-dimensional basis vectors as the codewords. Figure 3 shows the result from training neural classifiers with different configurations. The figure shows that the logistic activation outperformed the softmax in all the cases we tested.

We used the random coding method in algorithm 1 to search for bipolar codewords. We varied the value of M and searched over 10,000 iterations for the best code $\mathbf{C}^*$ with the minimum mean μ_c^*. Figure 2 displays different sets of bipolar random codewords from algorithm 1 with $p = 0.5$ and $K = 100$. The codewords came from the bipolar Boolean cube $\{-1, 1\}^M$. Figures 2a-2c show the respective bipolar codewords for code length 20, 60, and 100 using algorithm 1. Figure 2d shows the bipolar basis vector with $K = 100$ from $\{-1, 1\}^{100}$. Table II shows that decreasing the mean μ_c of code $\mathbf{C}$ increases the classification accuracy of the classifiers trained with the codewords. This is true when the length M of codewords is such that $M < K$. We also found the best set of codewords with $p = 0.5$. Figure 4 also supports this.

Table III shows that logistic networks can achieve high accuracy with small values of M. The table shows that the random codewords can achieve a comparable classification accuracy with a small code length M relative to the accuracy from training with the softmax output activation using K binary basis vectors from $\{0, 1\}^K$ as the codewords. It took

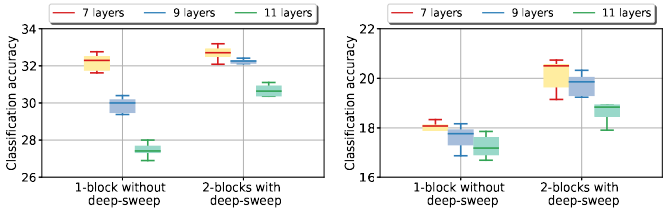


(a) CIFAR-100 (b) CIFAR-100 (c) Caltech-256 (d) Caltech-256

Fig. 5: Random bipolar coding and ordinary BP. Algorithm 1 picked K codewords from $\{-1, 1\}^M$. The marginal increase in classification accuracy with an increase in the code length M decreased as M approached K. (a) shows the classification accuracy of the deep neural classifiers trained with the random bipolar coding (aAgorithm 1). (b) shows the classification accuracy of the neural classifiers with 5 hidden layers. The accuracy increased by 8.31% with an increase from $M = 10$ to $M = 40$ and the accuracy increased by 0.61% with an increase from $M = 40$ to $M = 100$. (c) shows the classification accuracy of the deep neural classifiers trained with codewords generated with random bipolar coding. (d) shows the classification accuracy of neural classifiers with 5 hidden layers. The accuracy increased by 4.92% with an increase from $M = 10$ to $M = 80$ and the accuracy increased by 0.40% with an increase from $M = 80$ to $M = 200$.

TABLE IV: Deep-sweep versus ordinary backpropagation learning for deep neural classifiers and basis vectors as the codewords. We compared the effect of the algorithms on the classification accuracy of the classifiers. We used the bipolar basis vectors from $\{-1, 1\}^K$ as the codewords. Deep-sweep method outperformed the ordinary BP with deep neural classifiers. The deep-sweep benefit increased with an increase in the depth of the classifiers.

No. of Layers	No. of Blocks	Layers per Block	Deep-sweep	Classification Accuracy	
				CIFAR-100	*Caltech 256*
5 layers	1	5	**No**	32.65 ± 0.83%	19.06 ± 1.00%
	2	3	Yes	30.62 ± 0.41%	18.59 ± 0.66%
7 layers	1	7	No	32.19 ± 0.85%	17.93 ± 0.70%
	2	4	**Yes**	32.69 ± 0.70%	20.11 ± 1.22%
	3	3	Yes	27.95 ± 0.69%	16.25 ± 0.65%
9 layers	1	9	No	29.89 ± 0.78%	17.61 ± 0.91%
	2	5	**Yes**	32.20 ± 0.32%	19.75 ± 0.83%
11 layers	1	11	No	27.47 ± 0.78%	17.25 ± 0.85%
	2	6	**Yes**	30.68 ± 0.57%	18.77 ± 1.19%
13 layers	1	13	No	25.55 ± 1.09%	16.40 ± 0.57%
	3	5	**Yes**	30.76 ± 0.74%	18.47 ± 0.56%



(a) CIFAR-100 (b) Caltech-256

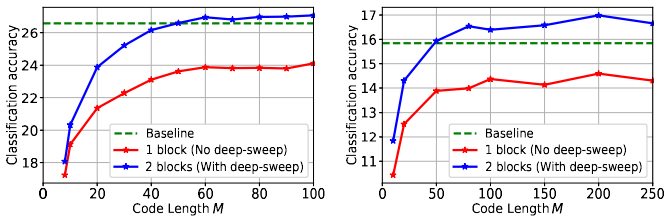
Fig. 6: Deep-sweep training method outperformed ordinary backpropagation. The deep neural classifiers used bipolar logistic functions for output activations. We used K bipolar basis vectors from the bipolar cube $\{-1, 1\}^K$ as the codewords with bipolar logistic outputs. We compared the effect of training with the deep-sweep method or ordinary backpropagation. Deep-sweep outperformed ordinary BP with deep networks. (a) shows the classification accuracy obtained from the classifiers with different sizes. (b) shows the classification accuracy obtained from the classifiers with different sizes.

TABLE V: Finding the best block size with the deep-sweep algorithm. We trained deep neural classifiers with the bipolar basis vectors from $\{-1, 1\}^K$ as the codewords. The relationship between the classification accuracy and the block size with a fixed number of blocks B follows an inverted U-shape.

No. of blocks	No. of layers per block	Classification Accuracy	
		CIFAR-100	*Caltech 256*
2 blocks	3 layers	30.62 ± 0.41%	18.59 ± 0.66%
	4 layers	32.69 ± 0.70%	20.11 ± 1.22%
	5 layers	32.20 ± 0.32%	19.75 ± 0.83%
	6 layers	30.68 ± 0.57%	18.77 ± 1.19%
3 blocks	3 layers	27.95 ± 0.69%	16.25 ± 0.64%
	4 layers	31.84 ± 0.84%	18.76 ± 0.60%
	5 layers	30.76 ± 0.74%	18.47 ± 0.61%
4 blocks	3 layers	27.94 ± 0.62%	13.96 ± 0.51%
	4 layers	28.45 ± 0.56%	16.97 ± 1.37%
	5 layers	25.80 ± 0.53%	16.64 ± 0.80%
5 blocks	3 layers	26.28 ± 0.88%	11.79 ± 1.02%
	4 layers	29.69 ± 1.12%	15.06 ± 0.89%
	5 layers	23.15 ± 1.86%	14.47 ± 1.00%

$M = 40 = 0.4K$ to get between $88\% - 90\%$ of the classification accuracy from using the softmax activation with $M = K = 100$ on the CIFAR-100 dataset. It took $M = 80 < 0.32K$ to get between $84\% - 101\%$ of the classification accuracy from using the softmax output activation (with $M = K = 256$) on the Caltech-256 dataset. The random codes with $M = 80$ outperformed the softmax activation with $M = 256$ for neural classifiers with 5 or 7 hidden layers. Figure 5 shows that the marginal increase in classification accuracy with an increase

in the code length M decreases as M approaches K.



(a) CIFAR-100 (b) Caltech-256

Fig. 7: Deep-sweep training with the random bipolar code search and $(M < K)$ outperformed the baseline. The baseline is training with the combination of ordinary BP and softmax activation with the binary basis vectors from $\{0,1\}^K$. We compared the effect of the deep-sweep method with code length M on the classification accuracy of deep neural classifiers. (a) shows the performance of deep neural classifiers with 9 hidden layers and trainied with the ordinary BP (no deep-sweep). It also show the perfromance of a 2-block netowrk with 5 hidden layers per block and trained with the deep-sweep method. (b) shows the performance of deep neural classifiers with 11 hidden layers and the ordinary BP (no deep-sweep). It also show the performance of a 2-block network with 6 hidden layers per block and trained with the deep-sweep method.

Table IV shows the benefit of training deep neural classifiers with the deep-sweep method in Algorithm 2. The deep-sweep training method reduces both the vanishing-gradient and slow-start problem. Simulations showed that the deep-sweep method improved the classification accuracy of deep neural classifiers. The deep-sweep benefit increases as the depth of the classifier increases. Figure 6 also shows that the deep-sweep method outperformed ordinary BP with deep neural classifiers. Table V shows the relationship between the accuracy and the block size with the deep-sweep method. The relationship follows an inverted U-shape with a fixed number of blocks B.

We also compared the effect of using the deep-sweep method and Algorithm 1 to pick the codewords. Figure 7 shows that the deep-sweep and random coding method with $M = 40 = 0.4K$ outperformed training with the 100 basis vectors as the codewords (with softmax output activation) without the deep-sweep. We used the CIFAR-100 dataset with $K = 100$ in this case. We also found the same trend with the models we trained on the Caltech-256 dataset. The combination of the deep-sweep and random coding method with $M = 80 < 0.32K$ outperformed training with basis vectors from $\{0,1\}^K$ as the codewords (with softmax output activation) with the ordinary BP.

IV. Conclusion

Logistic output neurons with random coding allow a given deep neural classifier to encode and accurately detect more patterns than a network with the same number of softmax output neurons. The logistic output layer of a neural block uses length-M code words with $\log_2 K \leq M < K$. Algorithm 1 gives a simple way to randomly pick K reasonably separated bipolar codewords with a small code length M. Many other algorithms may work as well or better. Each block has so few hidden layers that there was no problem of vanishing gradients. The network instead achieved depth by adding more blocks. Deep-sweep training further outperformed ordinary backpropagation with deep neural classifiers. Using bidirectional backpropagation [18]–[20] or proper noise-boosting [1], [2], [21], [22] should further improve deep-block behavior.

References

[1] O. Adigun and B. Kosko, "Noise-boosted bidirectional backpropagation and adversarial learning," *Neural Networks*, vol. 120, pp. 9–31, 2019.

[2] B. Kosko, K. Audkhasi, and O. Osoba, "Noise can speed backpropagation learning and deep bidirectional pretraining," *To appear in Neural Networks*, 2020.

[3] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.

[4] A. N. Gorban, I. Y. Tyukin, D. V. Prokhorov, and K. I. Sofeikov, "Approximation with random bases: Pro et contra," *Information Sciences*, vol. 364, pp. 129–145, 2016.

[5] P. Baldi and R. Vershynin, "The capacity of feedforward neural networks," *Neural networks*, vol. 116, pp. 288–311, 2019.

[6] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[7] K. Audhkhasi, O. Osoba, and B. Kosko, "Noise-enhanced convolutional neural networks," *Neural Networks*, vol. 78, pp. 15–23, 2016.

[8] B. Kosko, K. Audhkhasi, and O. Osoba, "Noise can speed backpropagation learning and deep bidirectional pretraining," *Neural Networks*, 2020.

[9] J. A. Gubner, *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.

[10] A. Leon-Garcia, "Probability, statistics, and random processes for electrical engineering," 2017.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[12] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei, "What does classifying more than 10,000 image categories tell us?" in *European conference on computer vision*. Springer, 2010, pp. 71–84.

[13] M. R. Gupta, S. Bengio, and J. Weston, "Training highly multiclass classifiers," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1461–1492, 2014.

[14] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[15] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[16] W. P. J., "Beyond regression: New tools for prediction and analysis in the behavioral sciences." *Doctoral Dissertation, Applied Mathematics, Harvard University, MA*, 1974.

[17] 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.

[18] O. Adigun and B. Kosko, "Bidirectional representation and backpropagation learning," in *International Joint Conference on Advances in Big Data Analytics*, 2016, pp. 3–9.

[19] O. Adigun and B. Kosko, "Bidirectional backpropagation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 5, pp. 1982–1994, 2019.

[20] O. Adigun and B. Kosko, "Training generative adversarial networks with bidirectional backpropagation," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1178–1185.

[21] O. Osoba and B. Kosko, "The noisy expectation-maximization algorithm for multiplicative noise injection," *Fluctuation and Noise Letters*, vol. 15, no. 01, p. 1650007, 2016.

[22] O. Adigun and B. Kosko, "Using noise to speed up video classification with recurrent backpropagation," in *International Joint Conference on Neural Networks*. IEEE, 2017, pp. 108–115.