Learning with Local Gradients at the Edge

Michael Lomnitz
Center for Vision Technologies
SRI International
Menlo Park, U.S.A.
michael.lomnitz@sri.com

Zachary Daniels
Center for Vision Technologies
SRI International
Princeton, U.S.A.
zachary.daniels@sri.com

Saurabh Farkya
Center for Vision Technologies
SRI International
Princeton, U.S.A.
saurabh.farkya@sri.com

Michael Isnardi
Center for Vision Technologies
SRI International
Princeton, U.S.A.
michael.isnardi@sri.com

David Zhang
Center for Vision Technologies
SRI International
Princeton, U.S.A.
david.zhang@sri.com

Michael Piacentino
Center for Vision Technologies
SRI International
Princeton, U.S.A.
michael.piacentino@sri.com

Abstract—To enable domain adaptation of AI on edge devices with fast convergence and low memory, we present a novel backpropagation-free optimization algorithm dubbed Target Projection Stochastic Gradient Descent (tpSGD). tpSGD uses layer-wise stochastic gradient descent (SGD) and local targets generated via random projections of the labels to train the network layer-by-layer with only forward passes. It doesn’t require retaining gradients during optimization, thus greatly reducing memory allocation compared to SGD backpropagation (BP) methods. Compared to other target projection methods, tpSGD generalizes the concept to arbitrary local layer-wise loss functions. Our method performs comparably to BP gradient-descent within ~5% accuracy on relatively shallow networks of fully connected layers, convolutional layers, transformers, and recurrent layers. tpSGD also outperforms other state-of-the-art gradient-free algorithms with competitive accuracy and less memory and compute time.

Index Terms—machine learning, deep learning, neural networks, backpropagation-free training, edge computing

I. INTRODUCTION

AI-based systems operating in rapidly changing environments need to adapt in real-time. The adaptability of current edge devices is limited by Size, Weight and Power (SwAP) constraints, the memory wall from current system architectures [14], and the lack of efficient on-device learning algorithms. As a result, current AI systems adapt by retraining on the cloud. Adaptation (domain transfer for a new data distribution) often requires fine-tuning of the entire or partial network. Similarly, it is often necessary to perform conversion from full precision to fewer bits [19] to fit the model within a limited hardware architecture, and doing so might require updating the network by distillation. Despite the fact that gradient-descent (GD) backpropagation (BP) is one of the most studied and successful approaches, the dominant approach still exhibits limitations, including vanishing and exploding gradients [18], difficulty with data quantization, and inability to handle non-differentiable parameters [13].

Extending GD BP to on-device training exhibits several additional problems related to limited memory availability: 1) The weight transport problem requires each layer to have full knowledge of other weights in the neural network (NN); 2) The update locking problem requires a full forward pass before the feedback pass. In GD BP, all layer inputs and activations in both forward and backward passes need to be buffered before weight updates are complete. To overcome issues associated with GD BP, especially for training at the edge, many gradient-free (GF) methods have been studied. Early GF approaches such as Genetic Algorithms [2], Swarm Optimization [2], Alternating Direction Method of Multipliers (ADMM) [3], etc. solved the learning problem without using GD. However, they do not show distinctive energy-time saving nor are they memory-efficient and scalable. Biologically-plausible approaches such as Hyperdimensional Computing [10] and Spiking NNs [23] exhibit energy and/or memory-efficient learning; nevertheless, training these models can be challenging, and working over binary represented weights and inputs generally requires approximation from floating point NNs via distillation. ADMM decomposes the network training into a sequence of sub-steps that can be solved as simple linear least-squares problems. ADMM optimizes each layer using the following layer as a condition. As such, it is a forward pass-only algorithm. ADMM variants (Scalable ADMM, dADMM, pdADMM [24]–[26]) have tried to efficiently utilize available hardware resources and reduce compute time. Although relatively mature and scalable to large problems, the learning time of ADMM is still much longer compared to other GF methods such as KPnet [30].

Most of the GF approaches have demonstrated their performance in shallow NN models and in fully connected layers. They are less scalable and produce less accurate results than deep NN models trained with BP. However, the combination of these approaches shows a promising trend. KPnet combines Extreme Learning Machines (ELM) [9] with label encoding and target propagation to speed up training and improve scalability. LM-KARnet [29] trains with time...
savings of 10-100X in shallow networks. ZORB \[20\] combines rapid training of ELMs and the Moore-Penrose (MP) pseudoinverse. It is 30X faster than the Adam optimizer on shallow networks and also achieves comparable accuracy.

Frenkel et al. proposed Direct Random Target Projection (DRTP) \[5\], \[17\], a method introducing target projection into GD learning algorithms. DRTP alleviates two key BP issues (weight transport problem and update lock) by enabling each layer to be updated with local information as the forward evaluation proceeds and was proven to work on shallow networks. DRTP allows training hidden layers at low computational and memory costs, and the paper showed that training event-based CNN processor via DRTP \[6\] required only 16.8\% power and 11.8\% silicon area overhead compared to a standard training approach. Based on this concept, we propose a novel BP-free optimization algorithm called target projection Stochastic Gradient Descent (tpSGD), capable of training NN models at the edge. tpSGD trains NNs layer-by-layer using only forward passes and extends target projection to arbitrary local layer-wise loss functions. tpSGD has the following unique features:

- Local efficient computation per layer using SGD and target projection. Our method performs comparably to GD BP (within 1\% accuracy) on relatively shallow networks with 1-6 trainable convolutional and/or fully connected layers, while eliminating the need to retain gradients and compute through the network in the backward pass.
- Extends target projection to multichannel convolutional layers via filter-based sampling, enabling CNN training.
- Extends target projection to recurrent neural networks (RNN) (standard, stacked multi-layer RNNs, and bidirectional RNNs) performing within 5-10\% accuracy of BP-trained networks on several benchmark datasets.
- Extends target projection to transformer-based neural networks with similar accuracy and lower memory usage.
- Demonstrates state-of-the-art (SoA) performance and decreased training time compared to other BP-free algorithms (e.g., dIADMM, ZORB, KARnet).

II. METHODOLOGY

A. Target Projection and Target Propagation

In order to remove backpropagation and still train a neural network, we need to establish a mechanism to provide a signal to the intermediate layers. In target propagation \[20\], labels are “propagated” backward through the network by sequentially inverting each operation in order to obtain the corresponding output to the labels. In contrast, in target projection \[5\], \[30\], one-hot encodings of the labels are directly projected to a given layer during the optimization step. Given an intermediate layer \(L_i\) in a NN, we generate local targets \(y_i\) for the layer by projecting the data labels \(y^*\) via a random projection matrix \(P_i\). By using target projection instead of propagation, tpSGD replaces the need to invert all layers and activations with a single matrix multiplication.

B. Target Projection SGD

tpSGD is designed with the goal of scaling BP-free training to larger datasets and deeper networks required for complex tasks. Unlike ZORB \[20\] or LM-KARnet \[29\], we utilize GD-based optimization for the individual layers instead of the pseudo-inverse which optimizes weights on an entire dataset at once. While this limits ZORB and LM-KARnet even on relatively small datasets, such as CIFAR, our algorithm can process data in batches providing more freedom to extend to datasets and tasks with larger memory requirements. In contrast to BP, which calculates a full forward and backward pass through the network, tpSGD is a feedforward-only algorithm. In tpSGD, we train each layer in the network sequentially starting from the layers closest to the input. For a given layer \(L_i\) in a NN with \(i \in N\) layers, the input to that layer \(x_i\) is obtained by running the forward pass over all previous \(j = 1\) to \(i - 1\) layers. The target output \(y_i\) is obtained via a random projection of the one-hot encoding of the data labels. The input \(x_i\) and projected targets \(y_i\) are used to train the layer using the Adam optimizer and the Mean Squared Error (MSE) between the predictions and \(y_i\). As the layers are each trained with local gradients, this eliminates the need to retain the entire graph during training and resolves the issue of vanishing/exploding gradients. Once a layer is trained, we fix the weights and move on to the next layer following the same approach until the final layer is reached as we no longer require a projection.

C. Connections to Existing Projection-Based Training

tpSGD shares close connections with a few prominent BP-free, projection-based training algorithms. We highlight connections between Direct Feedback Alignment (DFA) \[17\], Error-Sign-Based Direct Feedback Alignment (sDFA) \[5\], Direct Random Target Projection (DRTP) \[5\], Deep Supervised Learning Using Local Errors \[16\], and our novel tpSGD approach (tpSGD using layer-wise \(\ell_1\)-error (tpSGD\(_{\ell_1}\)) and \(\ell_2\)-error (tpSGD\(_{\ell_2}\))). In Table I, we inspect the optimization objectives of the aforementioned algorithms and investigate how this affects the weight update steps at layer \(i\). Consider a simple linear layer \(i\) with non-linear activation function: \(y_i = \sigma_i(z_i) = \sigma_i(x_i W_i)\). \(J(\cdot)\) is a loss function, \(x_i\) and \(y_i\) are the input and output of layer \(i\), respectively. \(W_i\) are the weights associated with layer \(i\) and \(\delta y_i\) is the estimated loss gradients for the outputs of layer \(i\). \(y_N\) is the predicted output of the final layer, \(y^*\) is the ground truth one-hot encoding of the labels and \(P_f\) is the layer-dependent projection matrix of the targets. \(M_i\) is a layer-dependent linear classifier, and \(s_i\) is the output of this classifier. \(\eta\) is the learning rate (including multiplicative constants). \(\odot\) is the element-wise multiplication operator.

DFA \[17\], sDFA \[5\], and DRTP \[5\] all attempt to minimize a global objective (cross-entropy between the ground truth labels and final layer output) using noisy gradient steps.
TABLE I: Summary of connections between tpSGD and existing projection based training

| Method       | $J_i(.)$                                      | $W^{(t+1)}$ |
|--------------|----------------------------------------------|-------------|
| DFA [17]     | Cross-Entropy$(y^*, y_N)$                    | $W_i^{(t)} + \eta x_i^T (P_i (y^* - y_N) \odot \sigma_i'(z_i))$ |
| sDFA [5]     | Cross-Entropy$(y^*, y_N)$                    | $W_i^{(t)} + \eta x_i^T (P_i \text{sign}(y^* - y_N) \odot \sigma_i'(z_i))$ |
| DRTP [5]     | Cross-Entropy$(y^*, y_N)$                    | $W_i^{(t)} + \eta x_i^T (P_i y_i^* \odot \sigma_i'(z_i))$ |
| Local Errors [16] | $\ell(y^*, s_i)$ | $W_i^{(t)} - \eta x_i^T (M_i^T \delta y_i^* \odot \sigma_i'(z_i))$ |
| tpSGD$_{\ell1}$ | $||P_i y^* - y_i||_1$                     | $W_i^{(t)} + \eta x_i^T (\text{sign}(P_i y^* - y_i) \odot \sigma_i'(z_i))$ |
| tpSGD$_{\ell2}$ | $||P_i y^* - y_i||_2$                     | $W_i^{(t)} + \eta x_i^T ((P_i y^* - y_i) \odot \sigma_i'(z_i))$ |

DFA and sDFA require a complete forward pass through the network and then compute estimated loss gradients for a given layer based on random projections of the gradients of the loss at the final layer. sDFA uses only the sign of the loss gradients whereas DFA uses the full magnitude of the loss gradients. Since both of the methods require a full pass through the network, they are not compatible with tpSGD. DRTP’s loss gradients are a surrogate of those of sDFA. DRTP decouples the estimated loss gradients from the final layer predictions. To obtain layer-wise gradients w.r.t. the global loss, DRTP uses a random projection matrix to project one-hot labels as the estimated loss gradients for the global objective. DRTP does not require a full forward pass before updating the weights, and DRTP can update layers independent of one another, making DRTP fully compatible with tpSGD.

In contrast to other approaches, tpSGD minimizes a collection of local losses, forcing layer outputs to align with random projections of the one-hot ground truth labels (serving as targets for optimization). tpSGD is similar to Mostafa et al.’s work [16], which also showed effective local layer-wise-training. Mostafa et al. projects the layer outputs to the target space via random auxiliary classifiers instead of projecting the targets to the layer through a random projection matrix.

tpSGD is compatible with any generic layer-wise loss between the layer-wise targets and layer-wise outputs, but we explicitly consider the $\ell_1$-norm and $\ell_2$-norm for this paper.

D. Layer Projection Definitions

In tpSGD, we provide learning targets for intermediate layers by projecting the ground truth labels via randomly generated matrices. This section discusses the projection strategies for linear, convolutional, recurrent, and transformer layers.

1) Linear Layer Projections: In the simplest case of a fully connected layer with $n$-nodes and a classification problem with $bs$ batch size, $nc$ classes, we map a batch of labels with dimensions $(bs, nc)$ to one with $(bs, n)$ by multiplying by a random matrix $P$ with dimensions $(nc, n)$. We tested sampling from different distribution functions (uniform, normal, etc.), but we found no noticeable difference in performance.

2) Conv2D Layer Projections: Next we extend the approach to CNNs. Assume the output of a given Conv2D layer has dimensions $(bs, nx, ny, nf)$, where $bs$ is the size of the mini-batch being processed, $nx$ and $ny$ are the respective dimensions of the filtered image, and $nf$ are the number of filters. In the naive approach, we generate a single, long projection matrix $P$ with dimensions $(nc, nx \times ny \times nf)$ and reshape the output to match the target dimensions. When extending target projection via random matrices for CNNs, we observed that sampling the target features from a single distribution did not provide additional performance as the number of filters in a Conv2D layer was increased. Instead, we propose a filter-based sampling approach. We generate $nf$ different projection matrices $P_i$ for each of the $i \in \{1, 2, ..., nf\}$ filters, sampling from a normal distribution with varying standard deviations in the range $[0, 1]$ at equally spaced intervals $\sigma_i = i/nf$.

3) Recurrent Layer Projections: Past work [5] showed that target projection can train feedforward networks. To the best of our knowledge, we demonstrate for the first time that random target projection promotes learning in recurrent networks.

We formulate a simple recurrent cell:

$$H_{i+1} = \text{sigmoid}(H_i W_H + b_H + X_i W_X + b_X)$$

where $H_i$ is the hidden state input at time $t$, $X_i$ are the input features at time $t$, $W_H, b_H, W_X, \text{and } b_X$ are the learnable parameters of the cell (shared over all time steps), and $H_{i+1}$ is the hidden state that serves as input to the next time step.

In order to apply target projection to this recurrent cell, similar to BP through time [28], we unroll the recurrent cell over time for a fixed problem-specific sequence length. This produces a feedforward-like network except every layer shares a common set of learnable parameters and uses the hidden state output by the previous layer with time dependent input features. As such, in the unrolled recurrent layer there are two linear functions that feed into the non-linearity per unrolled “layer”. We can make use of the fact that we can freeze the weights of the recurrent cell for the current training iteration and update over all time steps simultaneously:

$$\begin{bmatrix}
H_1 \\
H_2 \\
\vdots \\
H_{end}
\end{bmatrix} = \sigma \left( \begin{bmatrix}
H_0 X_0 \\
H_1 X_1 \\
\vdots \\
H_N X_N
\end{bmatrix} \left( W_H \oplus \left(b_H + b_X\right) \right) \right)$$

where $\oplus$ represents row-wise addition of a vector to a matrix and $\sigma$ is the nonlinearity function (sigmoid function).
Fig. 1: Illustration of a linearized transformer encoder layer, trained as a single block using tpSGD.

Formulating the updates in this way, enables the $H_s$ to be treated as independent inputs, resulting in a recurrent layer that is equivalent to a feedforward linear layer plus nonlinear activation, which can be solved without BP via tpSGD. This formulation allows gradients to be computed over each $(H_t, X_t)$ separately and accumulated over all time steps as a standard average gradient matrix for all shared weights, significantly reducing memory requirements compared to BP.

The initial weights of the recurrent cells can be all zeros or drawn uniformly randomly with small magnitude. To initialize the projection matrices for tpSGD, we use random binary matrices with approximately orthogonal rows. To initialize the projection matrices for DRTP, we use random matrices with orthogonal rows. In both cases, we use different projection matrices for every time step.

E. Transformer Projections

We employ two important optimizations to extend our approach and enable edge training of attention-based neural networks. Figure 1 illustrates a standard definition of a transformer encoder layer, where the multi-headed self-attention layer is discussed in [27], allowing us to compress the sequence length and reduce both the memory footprint and inference time. Then, we train the entire transformer encoder layer as a whole, constructing the projection matrices as was done for linear layers to generate the target features during training with tpSGD.

III. EXPERIMENTS AND RESULTS

We compare tpSGD to BP for classification tasks on several datasets with different combinations of layer types. We also compare to several BP-free algorithms designed specifically to work at the edge. Both ZORB and LM-KARNet ingest an entire dataset in a single step during optimization. They excel in training with small datasets, but fail to extend to large datasets, deep NNs and complex tasks. We initially included dADMM and related algorithms in our benchmarking studies, but found them to exhibit substantially longer convergence times, so we excluded these comparisons.

A. Shallow Linear and Convolutional Networks

We begin by studying shallow networks on MNIST [4] and CIFAR [11]. Table II shows a comparison between the proposed tpSGD, ZORB, LM-KARNet, and the BP benchmark on MNIST using a 2-layer MLP with leaky ReLU activation. We report the mean value and standard deviation in parentheses over 25 random restarts for the training time and model accuracy. All three algorithms were benchmarked using implementations in TensorFlow 2.9 running on a relatively large NVIDIA RTX A5000 GPU to perform multiple training cycles to gather statistics in a reasonable amount of time. While tpSGD’s performance is 1-2% below both ZORB and LM-KARNet, we see sizable time savings for tpSGD. While both ZORB and LM-KARNet use the MP inverse to obtain the new weights, LM-KARNet replaces target propagation in ZORB with random projection. This accounts for the substantial speed up when comparing their training times: ZORB inverts each layer starting from the output while LM-KARNet projects using a single random matrix. However, LM-KARNet still uses the MP inverse to optimize each layer’s weights. Replacing this expensive operation in tpSGD leads to further decreases in the training time with near negligible loss in accuracy. tpSGD also yields substantial reduction in memory use: >10x in the case of ZORB and LM-KARNet and >3x compared to the BP baseline.

As a proof of concept deploying the algorithm to an edge solution, we reproduce the studies shown in Table III on an Nvidia Nano 4GB module with a maxwell GPU. We show the results obtained from a single training run.

In Table IV, we compare the performance (mean and std. deviation per metric) of a shallow model consisting of two Conv2D layers with leaky ReLU activations, and one final Linear layer on MNIST and CIFAR-10 using the same settings described earlier. The results show that tpSGD can nearly match the accuracy (within 1%) and training time with BP.

### TABLE II: Comparison of gradient free approaches with a 2-layer MLP trained MNIST using A-5000 GPU

| Algorithm | MNIST Test accuracy (%) | Training Time (s) | Peak mem. (GiB) |
|-----------|-------------------------|------------------|-----------------|
| BP        | 95.1                    | 2.1 (0.01)       | 0.58            |
| ZORB      | 89.4(0.22)              | 7.5(0.39)        | 1.74            |
| LM-KARNet | 89.9(0.12)              | 4.9(0.38)        | 1.8             |
| tpSGD     | 88.3(0.25)              | 1.7(0.01)        | 0.174           |

### TABLE III: Comparison of gradient free approaches with a 2-layer MLP trained on MNIST using an NVIDIA Nano embedded device

| Algorithm | MNIST Test accuracy (%) | Training Time (s) | Peak mem. (GiB) |
|-----------|-------------------------|------------------|-----------------|
| BP        | 90.2                    | 17.7             | 1.69            |
| ZORB      | 91.3                    | 9.9              | 1.93            |
| LM-KARNet | 89.3                    | 2.5              | 0.27            |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE IV: Comparison between tpSGD and BP on training a shallow CNN on MNIST and CIFAR-10

|         | tpSGD |         | BP     |
|---------|-------|---------|--------|
|         | Acc. (%) | Time(s) | Acc. (%) | Time(s) |
| MNIST   | 96.6(0.2) | 2.7(0.1) | 97.1(0.2) | 4.6(0.1) |
| CIFAR   | 46.5(1.2)  | 4.6(0.1)  | 46.8(1.1) | 4.8(0.4)  |

B. Time Series Modeling Using Recurrent Neural Networks

We also conducted experiments over a wide range of datasets ([1], [4], [7], [12], [22]) with differing properties (modality, number of training samples, number of classes, sequence length, number of features, feature data type, range of feature values, sparsity) to validate the training of the recurrent layers in combination with linear classifiers/multi-layered perceptrons for the classification of time series data. In the case of the UCF101 and “Twitter Sentiment Analysis - Word2Vec Encoding” datasets, we use a fixed external model to first preprocess the data into feature vectors. All other datasets work over the raw representations.

We focus on the case of a 1-layer RNN in combination with a linear classifier in Table V. We also validated that the approach works for a 2-layer stacked RNN [8] in combination with a 2-layer perceptron and a 1-layer Bidirectional RNN [21] in combination with a linear classifier. The results for these additional settings are similar to the single-layer RNN with linear classifier and are not reported for brevity’s sake.

In our main experiment, we use a hidden vector size of 512, a batch size of 32, and train for the equivalent of 20 epochs (backpropagation trains for 20 epochs, target projection methods train each layer for 10 epochs in order to maintain similar training time). We run ten trials with different seeds per setting and record accuracy, memory usage compared to the backpropagation model, and execution time compared to the backpropagation model. All models are trained/tested using tensorflow in eager execution mode.

We compare against random chance, a completely random network of equivalent structure, a network where the RNN is random, but the classifier is trained, and a structurally-equivalent model trained only using backpropagation. We also demonstrate that the method effectively trains with either DRTP or tpSGD_f1 as the optimizer for the recurrent layers, and performance is generally similar.

We verify that there is non-trivial learning of the recurrent layer(s) by comparing the model with randomly weighted recurrent layers and a trained classifier. We see significant improvement in all cases except the Seizure Activity Recognition dataset where even random features achieve relatively high performance. We also compare performance of the target projection-based models (column 5) to the performance of the gold-standard BP based models (column 6) to understand how well the model learns compared to an “upper bound” and similarly, compare to random chance (column 2) and models with random weights (column 3) to see how well the model learns compared to a “lower bound”. In all cases, the trained model significantly outperforms the completely random models and performs reasonably well compared to backpropagation.

The strength of tpSGD (low memory usage compared to backpropagation) is clearly seen from looking at Table V. Memory savings are related to size of the input feature vectors and length of the sequence. At peak memory usage, naïve implementations of BP through time store the gradients for all weights for all time steps. Our implementation of tpSGD stores a single gradient matrix for each shared set of weights averaged over all time steps, leading to reduced memory overhead. In the most extreme case, BP takes roughly 4.3 GB of VRAM at peak usage whereas tpSGD only takes 24 MB of VRAM at peak usage.

C. Time Series Modelling with Linearized Self-Attention

We further demonstrate the performance of tpSGD using the same time series datasets discussed in III-B for transformer based neural networks trained with tpSGD. Figure 2 compares a traditional, 6 layer transformer encoder trained with BP (green), a linearized transformer (linformer) with projection dimension p = 128 trained with BP (blue), and the same linformer trained with tpSGD (yellow). From left to right, the results show the average accuracy, training time and peak memory use obtained over 25 random restarts. The linformer trained with tpSGD obtains the highest accuracy on most datasets while reducing the peak memory by up to half and the training time by two thirds.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we explored a novel BP-free, feedforward-only optimization algorithm designed to enable training in resource-constrained environments, such as on edge devices. We discussed the connections between tpSGD and other existing BP-free algorithms and compared their performance when training in architectures such as MLPs, CNNs, RNNs, and Transformers. We found that tpSGD in training performs comparably to the BP SGD and BP-free algorithms in shallow MLPs, CNNs, RNNs, and Transformers and is superior to other BP-free algorithms in terms of memory and time. In applications at the edge, tpSGD is effective for fine-tuning adaptor layers in combination with fixed pre-trained feature extraction backbones as a means of facilitating adaptation to new domains. For CNN projections, we will explore whether we can learn the distribution of layer-wise target projections instead of through blind estimation/trial-and-error.

ACKNOWLEDGEMENTS

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via Contract No: 2022-21100600001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.
TABLE V: Results of training a single-layer RNN with linear classifier. Numbers in parentheses are standard deviations.

| Dataset          | Accuracy | Comp. Time Ratio | Memory Ratio | Accuracy | Comp. Time Ratio | Memory Ratio | Accuracy | Comp. Time Ratio | Memory Ratio |
|------------------|----------|------------------|--------------|----------|------------------|--------------|----------|------------------|--------------|
| Trained RNN (tpSGD) | 98.87 (0.07) | 0.889 (0.006) | 0.944 (0.005) | 97.63 (0.07) | 0.887 (0.007) | 0.964 (0.008) | 97.4 (0.08) |
| Trained RNN (BP)  | 82.15 (0.87) | 0.924 (0.01) | 0.94 (0.013) | 76.17 (1.12) | 0.895 (0.013) | 0.949 (0.014) | 74.92 (0.014) |

Fig. 2: Comparing a traditional 6 layer transformer encoder trained with BP (green), a linearized transformer (linformer) with projection dimension \( p = 128 \) trained with BP (blue), and the same linformer trained with tpSGD (yellow) in terms of accuracy, training time, and memory usage.

REFERENCES

[1] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. Physical Review E, 2001.

[2] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. Information sciences, 2013.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning, 2011.

[4] Li Deng. The mnist database of handwritten digit images for machine learning research. IEEE SPM, 2012.

[5] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. Frontiers in Neuroscience, 15, 2021.

[6] Charlotte Frenkel, Jean-Didier Legat, and David Bol. A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas, 2020.

[7] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. Technical Report, 2009.

[8] Salah Hiihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. NeurIPS, 8, 1995.

[9] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines: a survey. International journal of machine learning and cybernetics, 2011.

[10] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. Cognitive computation, 2009.

[11] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[12] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. NeurIPS, 2016.

[13] Stijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred Hero, and Pramod K. Varshney. A primer on zeroth-order optimization in signal processing and machine learning, 2020.

[14] Sally A. McKee and Robert W. Wissewski. Memory Wall. Springer, 2011.

[15] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. arXiv, 2017.

[16] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. Frontiers in neuroscience, 2018.

[17] Aríld Nøkland. Direct feedback alignment provides learning in deep neural networks, 2016.

[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012.

[19] Aswin Raghavan, Mohamed Amer, Sek Chai, and Graham Taylor. Binet: Bit-regularized deep neural networks. arXiv, 2017.

[20] Varun Ranganathan and Alex Lewandowski. Zorb: A derivative-free backpropagation algorithm for neural networks. arXiv, 2020.

[21] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. IEEE TSP, 1997.

[22] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv, 2012.

[23] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Maquieilier, and Anthony Maida. Deep learning in spiking neural networks. Neural networks, 2019.

[24] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In ICML. PMLR, 2016.

[25] Junxiang Wang, Yuan Lan, and Armand Joulin. Indications of nonlinear dynamics in feedback: Fixed random learning signals allow for feedforward training of deep neural networks. Frontiers in Neuroscience, 2016.

[26] Junxiang Wang, Fuxun Yu, Xiang Chen, and Liang Zhao. Toward model parallelism for deep neural network based on gradient-free admm framework, 2020.

[27] Junxiang Wang, Fuxun Yu, Xiang Chen, and Liang Zhao. Admm for efficient deep learning with global convergence. In KDD, 2019.

[28] Paul J Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 1990.

[29] Huiping Zhuang, Zhiping Lin, and Kar-Ann Toh. Training a multilayer neural network with low-memory kernel-and-range projection. Journal of the Franklin Institute, 2020.

[30] Huiqing Zhuang, Zhiping Lin, and Kar-Ann Toh. Training multilayer neural networks analytically using kernel projection. In 2021 IEEE ISCAS, 2021.