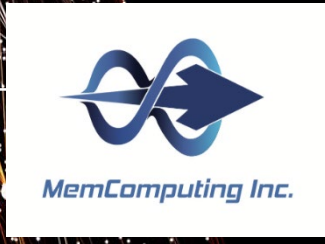
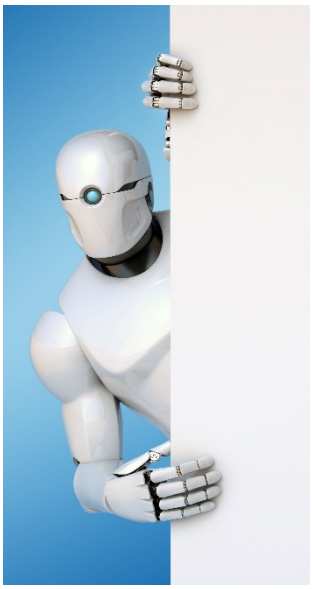
White Paper

An Efficient Approach to the Supervised Training of Deep Neural Networks using MemComputing

Breakthrough Computing Performance

Fabio L. Traversa
Erick Pederson





EXECUTIVE SUMMARY

Deep learning has been embraced by virtually every industry for decision support and even automated decision making. Deep Neural Networks (DNNs) support applications such as autonomous vehicles, image processing, financial transactions, language processing, aviation, manufacturing, fraud detection, and an almost endless list of additional applications. A DNN is only as good as its training.

Gradient-based methods represent the state of the art in the area of supervised training, where Adaptive Moment Estimation (ADAM) and Stochastic Gradient Descent (SGD) are among the most popular. However, these methods suffer from limitations and must commonly be augmented using a variety of algorithmic extensions, such as dropout, momentum, weight decay, and variable learning rate. In many cases, training a DNN is more of an art than a science.

We introduce a new supervised training method based on memcomputing technology to accelerate the training of DNNs. We show that the memcomputing-based approach is faster (better accuracy in fewer epochs) than ADAM & SGD with reduced error rates and provides more accurate DNNs. Memcomputing does not require additional algorithmic extensions that substantially reduces the number of hyperparameters to tune.

As a case study, we report the training of a fully connected feedforward DNN for the detection of Exotic High-Energy Particles and highlight the benefits of memcomputing versus ADAM and SGD.

1 Gradient-based training methods and their main limitations

Backpropagation (BP) was adopted in the mid-1980's as the preferred training algorithm because it was shown to be far faster than other commonly used methods. BP is a method of evaluating gradients through a computational graph. When that computational graph is a neural network, BP evaluates the gradients of an associated loss function, also known as a cost function, associated with the weights of the neural network. SGD and ADAM incorporate BP, and because of this, both suffer from some inherent limitations of backpropagation:

- The deeper the network (i.e., an increasing number of layers), the harder it becomes to train due to the well-known vanishing gradient effect [1], which slows down the propagation of the correction to hierarchically higher layers. Additionally, gradient updates from separate data points are calculated independently and frequently act antagonistically, slowing convergence. Adaptive learning rate extensions (e.g., ADAM, Adadelta, RMSprop) were introduced to limit this effect.

- It is still under intense debate as to what kind of local minima these methods converge to, but many times they converge to "bad" local minima leading to overfitting and a high error rate [2]. For this reason, many hot fixes such as dropout, weight decay, and a dynamic learning rate have been introduced. The challenge is that these are not necessarily guaranteed to solve these limitations, and lead to many extra hyperparameters that require additional tuning, implying an additional heavy calculation burden.

- Using mini-batches of data allows for a more efficient numerical calculation because the matrix-matrix multiplication algorithm can be leveraged in both CPU and GPU implementations [3]. However, utilizing larger mini-batches can slow down the convergence since the effects of different data points can cancel one another while averaging over the gradients. Therefore, a tradeoff between minibatch size and convergence rate must be found.

[1] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001
[2] H. Li. Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems, December 6391–6401, 2018
[3] G. Hinton, N Srivastava, K. Swersky, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, 2012

2 The Memcomputing Innovative Training Approach

The memcomputing approach aims towards a more global and parallelized optimization algorithm. Each datapoint contributes towards the network update concurrently and in parallel with the others, resulting in faster and more robust learning.

The memcomputing routine is an alternative to gradient descent-based methods. It is not based on minimizing the loss function of the network by means of updates of the learning parameters in the direction forecasted by the gradient (or by any variant), but rather it associates a dynamical system to the DNN whose evolution naturally converges to equilibria that map zeros of the loss function.

The associated dynamical system is designed to have properties that guarantee convergence and speed, such us being point dissipative and having hyperbolic equilibria [4]. The dynamical system is designed starting from the concept of self-organizing gates [4-5] that are a possible realization of memcomputing machines [6-7] representing a completely new computing architecture that can be simulated on virtually any CPU or GPU, and ultimately integrated on an integrated circuit of its own. A self-organizing gate for the DNN is designed to embed the evaluation of the neural network along an element of the dataset. The gate is designed to have an equilibrium point that is a mapping of the zero of loss function related to the element. Therefore, given a dataset input for the DNN, the dynamical system is defined by the network of self-organizing logic gates, one for each element of the dataset, sharing a terminal for each trainable parameter.

This approach allows for the trainable parameters of the network to be optimized together in a truly parallelized fashion, allowing for the training of deeper networks without the need of most hot fixes used for DNN training.

For a detailed look at the implementation of the memcomputing approach, see Appendix I.

[4] F. L. Traversa, M. Di Ventra, Polynomial-time solution of prime factorization and NP-complete problems with digital memcomputing machines, Chaos: An Interdisciplinary Journal of Nonlinear Science 27, 023107, 2017
[5] F. L. Traversa, M. Di Ventra, Memcomputing integer linear programming, arXiv:1808.09999, 2018
[6] F. L. Traversa, M. Di Ventra, Universal memcomputing machines, IEEE trans. on neural networks and learning systems, 26, 2702, 2015
[7] M. Di Ventra, F. L. Traversa, Perspective: Memcomputing: Leveraging memory and physics to compute efficiently, Journal of Applied Physics, 123, 180901, 2018

3 Case Study: The challenges in the detection of Exotic High Energy Particles

Collisions in high-energy particle colliders represent a crucial source of exotic particle discoveries. Nevertheless, a vast majority of particle collisions do not produce any exotic particles. For instance, the Large Hadron Collider (LHC) at CERN produces approximately 600 million collisions per second, but roughly 300 of these collisions result in a Higgs boson. For this very reason, **a significant challenge in finding exotic particles lies in the capability to solve difficult signal-versus-background classification problems**. Machine-learning approaches represent a useful tool to support this differentiation.

'Shallow' machine learning models represent the foundation of standard approaches used to support such discoveries. However, the Shallow machine learning models encompass 2 main limitations:
1) A limited capacity to learn complex nonlinear functions of the inputs
2) A reliance on a meticulous search through manually constructed nonlinear features

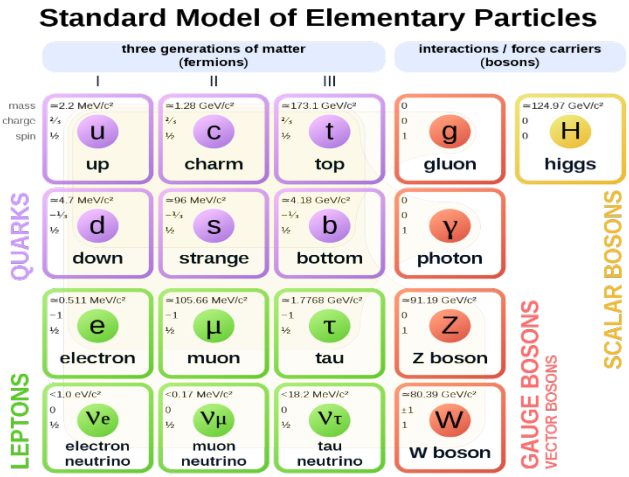
Recent advances in the field of deep learning make it possible to learn more complex functions and better discriminate between signal and background classes [8-9].

The memcomputing technology approach allows for particularly promising advances in this field, as shown by this recent project.

In this case study, we compare the most popular deep learning methods against a memcomputing technology approach.

[8] P. Baldi, P. Sadowski, D. Whiteson, Searching for Exotic Particles in High-Energy Physics with Deep Learning, Nature Communications, 5, Article number: 4308 (2014)
[9]Higgs Dataset used for the comparative testing: https://archive.ics.uci.edu/ml/datasets/HIGGS

The CERN Large Hadron Collider



4 Comparative Tests: Set up and Methods

NEURAL NETWORK (NN)

- **A fully connected feedforward network**
- **Composed of 5 layers of weights;**
  - 1 input layer of 28 nodes,
  - 4 hidden layers of 300 nodes each
  - A single node output layer.
- **Activation function** for hidden nodes: tanh
- **Output node utilization of a sigmoid function** to bound the output between [0,1].
- **Use of Binary Cross Entropy loss (BCE)** function, predominantly used for binary classification tasks.

DATA SELECTION

- **11,000,000 labeled input vectors** of length 28
  - 80% forming the training set,
  - 20% was set aside for testing.
- **The selection was picked at random.**

3 TRAINING METHODOLOGIES

- **ADAM** optimization algorithm implemented within PyTorch.
- **Stochastic Gradient Descent (SGD)** optimization algorithm implemented within PyTorch.
- **Custom algorithm based upon a memcomputing architecture.**

AUC-ROC: DEFINITION AND IMPORTANCE

For classification problems, performance measurement is based on the AUC - ROC Curve analysis. AUC - ROC curve represents a performance measurement for classification problem at various threshold settings. This curve indicates how accurate a model is to distinguish between classes.

- ROC Curve = Receiver Operating Characteristics curve = It is a probability curve. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0. In other words, it plots the false alarm rate versus the hit rate.
- AUC – Area Under the Curve. It represents a very good measure of separability.

→ **The ROC curve is useful for classification performance assessment for 2 main reasons**:

- ROC curves of different models can be easily compared for different thresholds.
- The area under the curve (AUC) can be used as a visual summary of the model skill.

→ **The higher the AUC, the better the model is at predicting the categories**

- AUC = 0.7: 70% chance that the model distinguishes between two different classes
- AUC = 0.5: the model has no discrimination capacity between two different classes
- AUC = nearly 0 : the model is actually reciprocating the classes
- In our case study, the higher the AUC, the better the model is at distinguishing between signal and background classification problems for the detection of exotic particles.

Source - Medium. (2020). *Understanding AUC - ROC Curve*. [online]
Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 [Accessed 2 Feb. 2020].

GLOSSARY

- AUC = Area under the ROC curve
- BCE = Binary Cross Entropy
- CPUs = Central Processing Units
- DMMs = Digital Memcomputing Machines
- DNNs = Deep Neural Networks
- NCNZF = non-convex nonlinear zero finding
- ROC = Receiver Operating Characteristic curve
- SGD = Stochastic Gradient Descent
- SOLGs = Self-Organizing Logic Gates

Method Set Up

HYPER-PARAMETER CONFIGURATIONS FOR MemCPU

- **Set of hyperparameters for the Memcomputing engine $th, \bar{r}, n$**
  - Defaults values were set as th = 0.035, $\bar{r}$ = 14, n = 8 as per a past project
  - These parameters are independent on the DNN type and dataset, they are related to the efficiency of the Memcomputing part of the algorithm therefore we can use the same values from past unrelated projects.
- **Best batch size selection using a modified bisection method.**
  - For the first epoch, a batch size is 350 was selected,
  - For the second epoch, a batch size of 700 was selected,
  - For all the other epochs, a batch size of 1700 was selected,
  - This batch size schedule led to the best convergence rates.
  - Batch sizes allow for GPU acceleration: utilizing the GPU implementation of our code within MATLAB, we found an 8X speed up with respect to the CPU. This can be substantially improved with a compiled implementation in CUDA and C++, rather than in an interpreter like MATLAB.
- **Other hyperparameters selection**
  - n = 6 instead of 8 could save some running time and preserve the convergence trend.
  - The $normmax$ parameter was evaluated by reintroducing lines 8, 9 and 10 of the algorithm, and an optimal value around $normmax$ = 1.5 was found

HYPER-PARAMETER CONFIGURATIONS FOR ADAM & SGD

Test were performed in order to provide the best possible performance

- **Batch Size**: Batch sizes of 64, 100, 200, 500, 1000, 5000, 10000 were tested. The best trade-off between speed of training and speed of convergence with a batch size of 1000. Below a batch size of 1000, no substantial speedup in training convergence was noticed. Using a batch size lower than 200 prevented us from utilizing GPUs for training, as anything less was faster using the CPU.
- **Loss Function**: Both MSE (Mean Squared Error) and BCE (Binary Cross Entropy) loss were tested and the BCE loss leads to faster convergence than MSE loss.
- **Weight initialization**: Both the uniform distribution initialization standard within the PyTorch Framework, as well as the Normal Distribution initialization were tested with no significant difference in their initialization scheme. The normal distribution was selected to remain consistent with previous publications on Memcomputing.
- **Dropout**: Values of 0.05, 0.1, 0.5 were tested. The convergence is slower using a dropout, without an appreciable increase in the final achieved AUC and in some cases a lower final AUC → our final tests were performed with no dropout. This is most likely due to the already noisy and large sized dataset along with batch size. Dropout in this case is less effective than it might be otherwise.
- **Weight decay**: Weight decay values of 1e-3, 1e-4, 1e-5, 1e-6 and 0 were tested. Weight decay improves the accuracy of the runs and a value of 1e-5 for SGD and for ADAM was selected
- **Learning Rate**: Learning rates of lr=0.01, 0.005, 0.001, 0.0001, as well as variable learning rates were tested. The static learning rates were unable to converge quickly, or at all. Learning rate schedulers were tested and reduced the learning rate by a factor f when the testing loss had not decreased by some threshold within n iterations. We tested f =0.999, 0.99, 0.9, 0.8, 0.5, 0.1, as well as n = 0, 1, 2, 3, 4, 5, 10. A reduction factor f = 0.999 with an epoch plateau of n = 1 represented the best balance between speed of convergence and final AUC. If n is set too low, the lr drops too quickly, while if n is set too high, the natural noise of the loss function will prevent the lr from ever decreasing within a reasonable number of epochs. We started with a lr=0.001 which was gradually reduced throughout training.
- **Optimizer:** SGD as well as ADAM were tested and a slightly faster convergence using ADAM was observed for the initial epoch. For larger number of epochs, SGD converges faster.

5

Performance Assessment
Memcomputing vs ADAM without Weight Decay

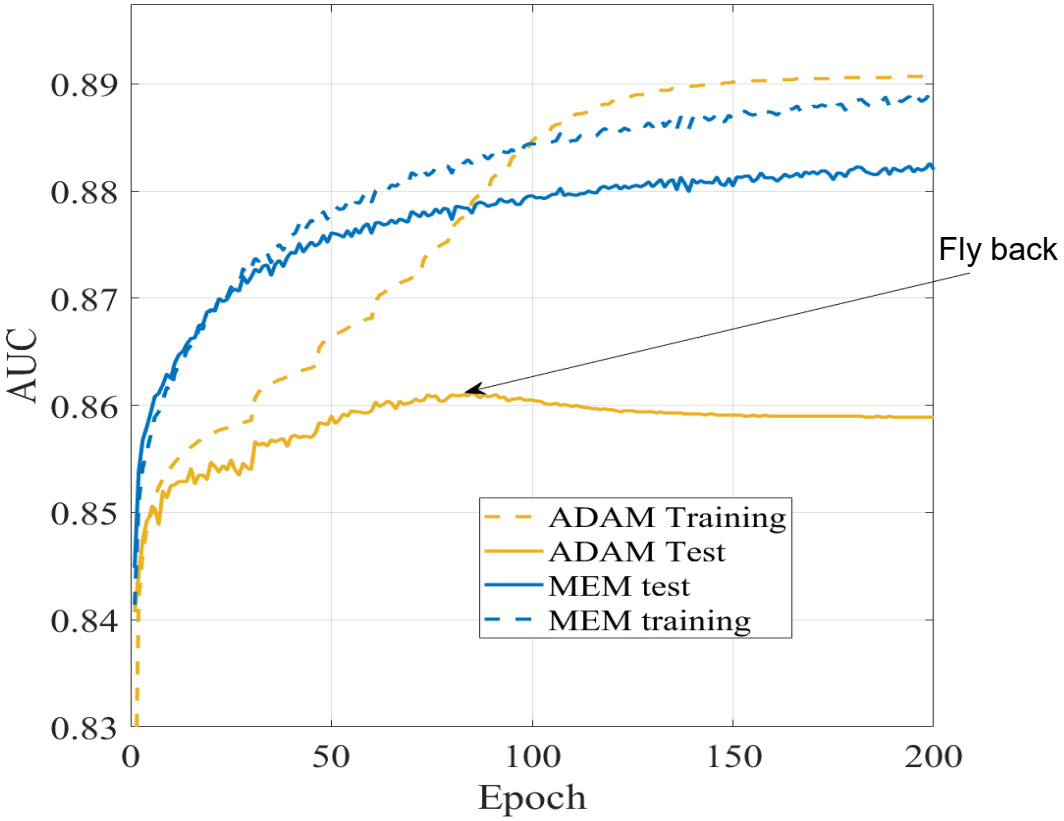


figure 1: Area under the ROC Curve (AUC) vs. Epochs

This first experiment shows a comparison between the ADAM method and the memcomputing approach. We intentionally removed the weight decay to show how hotfixes (the weight decay in his case) strongly affect the gradient-based training of the DNN. The Area Under the Curve (AUC in figure 1, see reference [8] for more details) is a quality measure for the training: the higher, the better. It represents a measure of the accuracy of the DNN in classifying inputs. The goal is to obtain the highest AUC in the shortest number of epochs. As you can see from figure 1, the memcomputing approach clearly shows the highest AUC.

Note the different curves labeled training and test. Training represents the known dataset that "teaches" the DNN. The test demonstrates how the DNN reacts to a random dataset that is not included in the training. The test indicates how one should expect the DNN to behave in production.

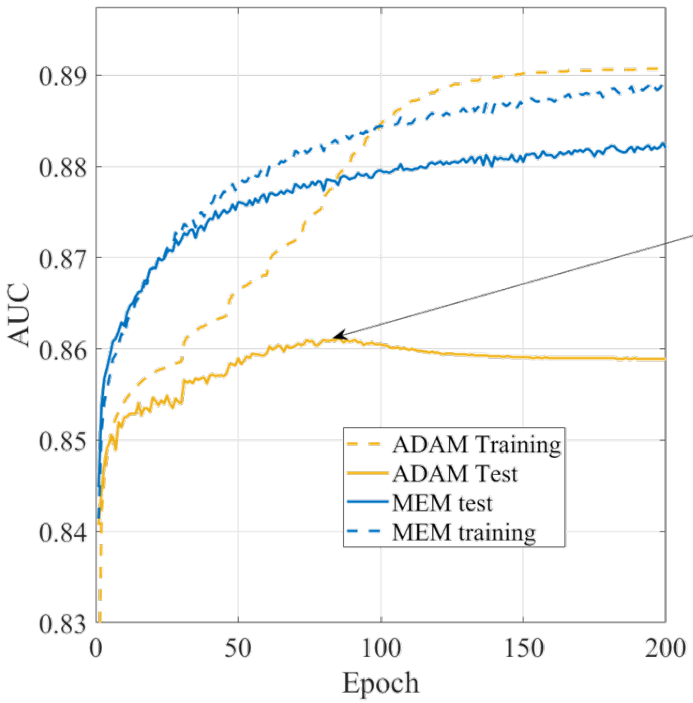


figure 1: Area under the ROC Curve (AUC) vs. Epochs

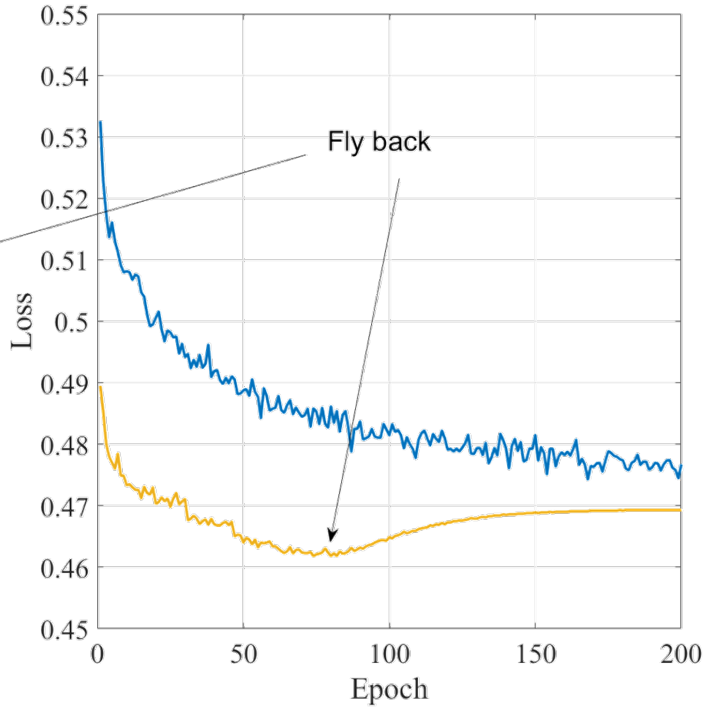


figure 2: Loss vs. Epochs.

Another measure used to analyze the training of a DNN is the loss function (figure 2).

- A lower loss function does not necessarily indicate better training because it depends on the local minima the training process approaches. There are cases (like in this work) where wider local minima can be shallower and therefore lead to higher loss function even if the accuracy of the training (defined here by the AUC) is better.
- Wider local minima imply a more robust model because slight variations in inputs remain within the minima. This allows the network to be more easily extended to datasets not included in the training set, for example, the testing set used in this work. If a training process approaches a very narrow local minima instead, there can be strong overfitting (represented by the gap between test and training AUC in figure 1) and in some cases the flyback effect, i.e., there is a critical value for the epochs for which the test AUC starts to decrease, and the loss function starts to increase (see the relationship between figure 1 and 2).

6 Performance Assessment Including the Weight Decay

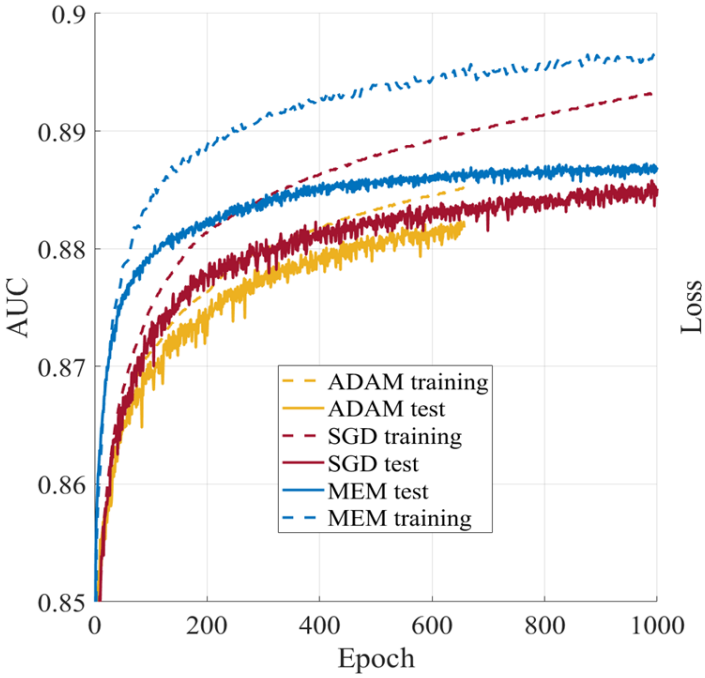


figure 3: Area under the ROC Curve (AUC) vs. Epochs

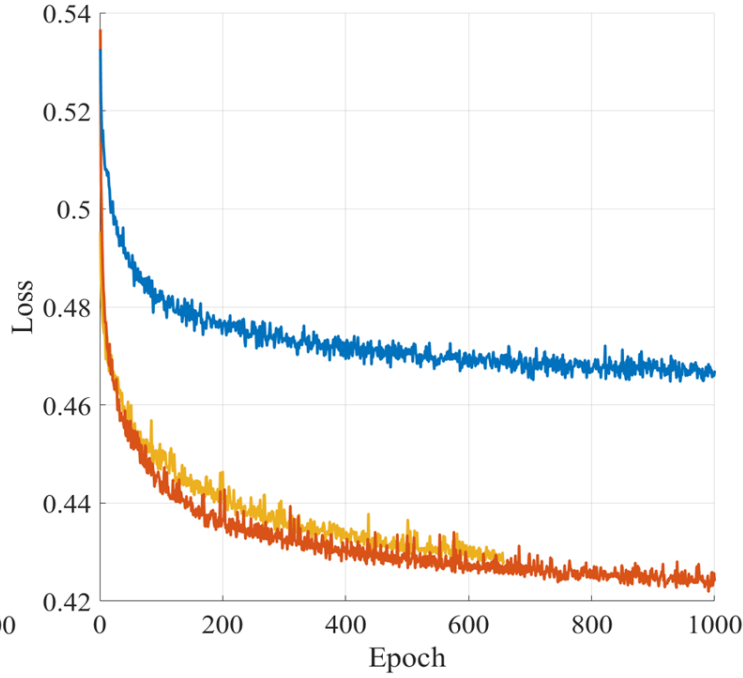


figure 4: Loss vs. Epochs.

In this second experiment, we include weight decay for both ADAM and SGD and train the DNN using ADAM, SGD, and memcomputing for a significantly larger number of epochs.

- Note specifically that the memcomputing approach does not require the addition of weight decay when performing training. Specifically, the weight decay helps training for the gradient-based approaches, limiting the overfitting and avoiding the flyback (see figure 3).
- However, including the weight decay has its drawbacks: it requires an extra hyperparameter to tune and slows down the convergence of the training.

Nevertheless, even including weight decay with ADAM and SGD, the training for memcomputing still shows superior performance (higher AUC in a shorter number of epochs) and similar overfitting (figure 3). Moreover, the loss function for memcomputing is still higher (figure 4), symptomatic of a different quality of training, and combined with higher AUC implies broader but shallower minima. That is, memcomputing provides higher accuracy for the testing dataset.

7 Performance Assessment - A Deeper Analysis

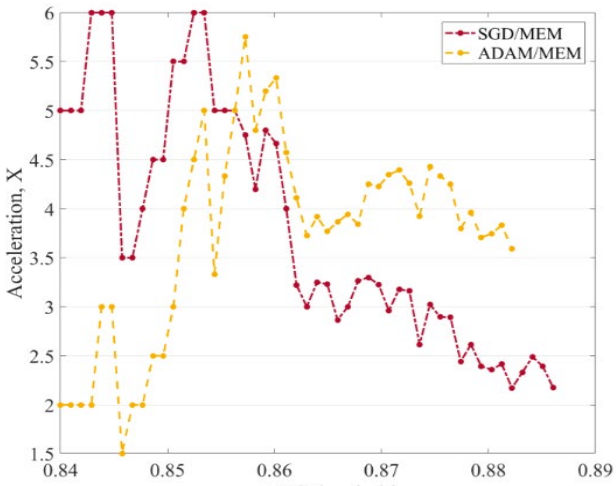


figure 5
Acceleration of MemCPU vs Adam and SGD for different AUC Thresholds

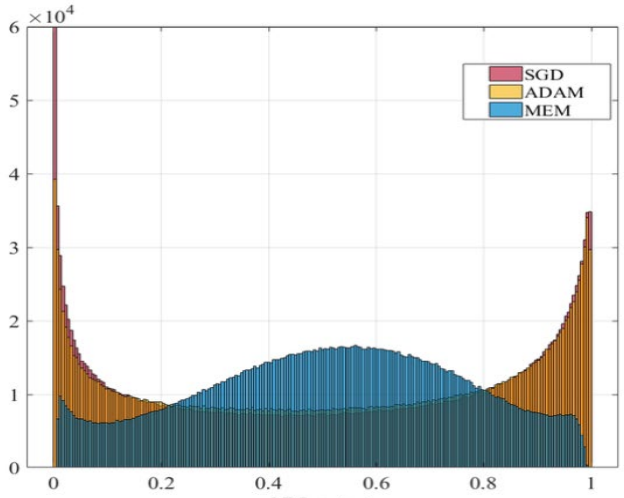


figure 6
Output distribution after 1,000 epochs

We report in figure 5 the acceleration provided by memcomputing compared against ADAM and SGD including weight decay. The acceleration is defined as the ratio:

- $\text{Acceleration} = \dfrac{\text{Nb epochs to reach a given AUC for method 1}}{\text{Nb epochs needed using the other method to reach the same AUC}}$

For a given accuracy defined by a threshold in the AUC reachable by the other methods, **memcomputing provides a speedup of up to 6X**. However, once the AUC exceeds 0.886, the gradient based methods break down, and therefore, the acceleration represented by memcomputing is virtually infinite. That is, you could train the DNN using ADAM and SGD forever, and it would never get beyond 88.6% accuracy.

Figure 6 reports the output distribution of the test set for the three methods.

- It is evident how both SGD and ADAM approach qualitatively similar distributions associated with similar minima for the DNN. The SGD and ADAM distributions are primarily on the boundaries, meaning that the network has learned to be extremely confident (overfit) for specific data-points, but fails to generalize to all data. This narrow minima fails to provide useful information from data that does not cleanly fall into the learned classifications, apparent from the steep fall-off from either end, as a small variation to the input can push the network out of the minima.
- On the other hand, **the memcomputing distribution is disbursed close to the center, showing that it tries to leverage as much information across the entire dataset.** This distribution allows for higher generalization outside of the original dataset. The shallower and wider minima allow a more significant variation in inputs while remaining within the minima to provide quality classification.

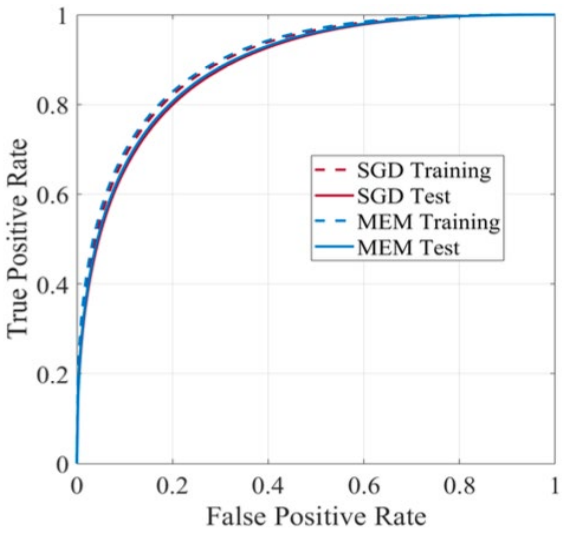


figure 7
ROC Curve

In figure 7, we report the Receiver Operating Characteristic (ROC) curve defined as the True Positive Rate (TPR) versus the False Positive Rate (FPR), i.e., the rate of correctly recognized exotic particles (TPR) versus the false positives. The AUC in the previous figures represents the area under the ROC curve. It is worth noticing that **the ROC curves for the representative gradient descent method (SGD) and memcomputing are qualitatively similar despite the different output distribution reported in figure 6.**

However, a closer look at the FPR and the TPR versus the threshold to declare an output "yes" or "no" (figure 8) clearly shows different shapes for the curves, a consequence of the different output distributions.

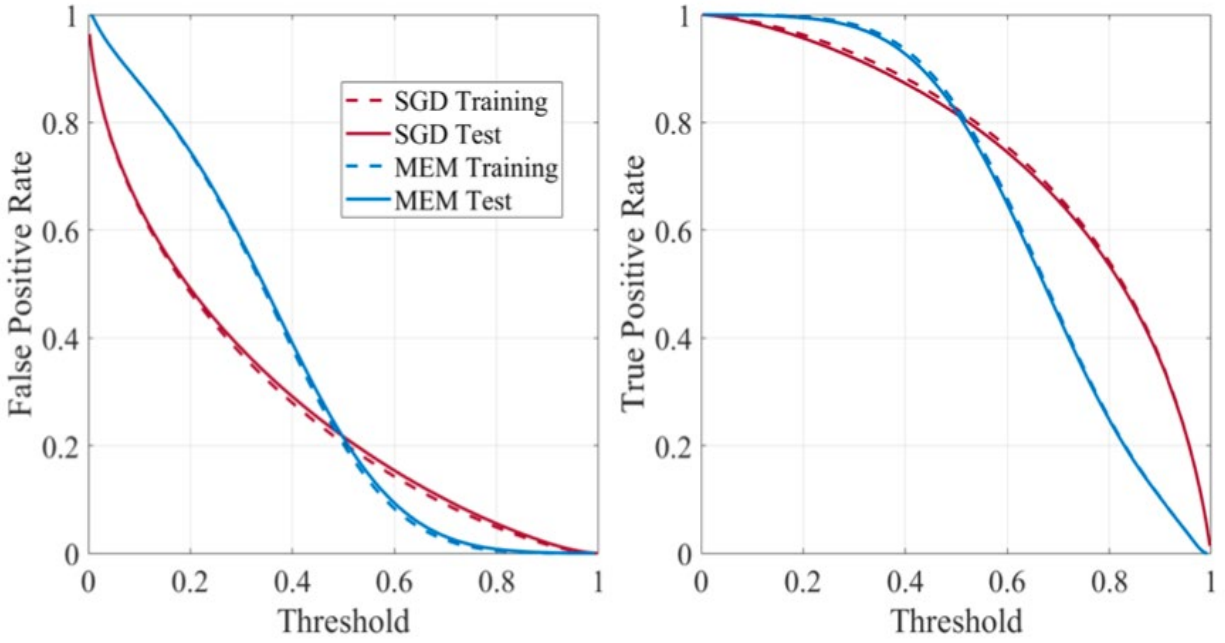


figure 8
True Positive and False positives rates

8 CONCLUSIONS

Memcomputing provides a novel and superior method for training neural networks. Our algorithm provides key improvements over the current industry standard of backpropagation. As a case study we reported performance on the detection of High-Energy Exotic particles problem.

- ☑ Faster convergence
- ☑ Higher accuracy
- ☑ More robust learning
- ☑ Fewer hyperparameters to tune
- ☑ Intrinsic lower overfitting
- ☑ Natural convergence to wider minima
- ☑ Stronger predictive and generalizable models

Memcomputing benefits go far beyond improved neural network training. It represents a paradigm shift in the way complex mathematical problems are solved!

- ☑ Cheaper, simpler, earlier than quantum computers
- ☑ Ultra-fast solutions, particularly for previously unsolved problems
- ☑ Emulated in software, runs on classical architecture
- ☑ A brand new/patented computer architecture
- ☑ Computation & memory combined in the same circuit
- ☑ Uses classical low power, low heat transistor technology

Please check our other White Papers and Case Studies!

MemComputing, Inc.'s disruptive coprocessor technology is accelerating the time to find feasible solutions to the most challenging operations research problems in all industries. Using physics principles, this novel software architecture is based on the logic and reasoning functions of the human brain.

MemComputing enables companies to analyze huge amounts of data and make informed decisions quickly, bringing efficiencies to areas of operations research such as Big Data analytics, scheduling of resources, routing of vehicles, network and cellular traffic, genetic assembly and sequencing, portfolio optimization, drug discovery and oil and gas exploration.

The company was formed by the inventors of MemComputing, PhD Physicists Fabio Traversa & Massimiliano Di Ventra and successful serial entrepreneur, John A. Beane.

MemComputing, Inc.
9909 Huennekens St.,
Suite 110
San Diego, CA 92121

http://www.memcpu.com
info@memcomputing.com

APPENDIX I
Implementation of the memcomputing approach

The Memcomputing routine output $M(\theta,Bi,n)$ is an approximation of the solution of the linear problem $G\Delta\theta = l - L(\theta, Bi)$ where G is the gradient of the function $L(\theta, Bi)$ with respect to θ. $L(\theta,Bi)$ represents the evaluation of the neural network for the batch Bi and the network parameters θ and l is the vector of labels associated to the batch Bi.

The Memcomputing approach builds a self-organizing circuit on the fly, designed to have an equilibrium point for θ that is a mapping of the solution of the linear system $G\theta = l$.

- The actual algorithm simulates the self-organizing circuit for just few time steps defined by n.
- The final configuration of the parameter increments $\Delta\theta$ is the output of the routine $M(\theta, Bi, n)$ used to update the learning parameters θ.

IMPLEMENTATION OF THE METHOD

1 $for\ e = 1;\ i++$
2 $\quad AUC_0(\theta,T) = AUC(\theta,T)$
3 $\quad for\ i = 1;\ i++$
4 $\quad\quad if\ mod(i,\bar{r}) = 1$
5 $\quad\quad\quad \Delta\theta = 0$
6 $\quad\quad \Delta\theta = \Delta\theta + M(\theta, B_i, n)$
7 $\quad\quad \theta = \theta + min\left((\Delta\theta, -th), th\right)$
8 $\quad\quad for\ j = 1;\ j < n_{layer};\ j++$
9 $\quad\quad\quad for\ h = 1;\ h < s^j_{input};\ h++$
10 $\quad\quad\quad\quad w_h^j = w_h^j / max\left(1, \left\|w_h^j\right\| / normmax\right)$

KEY DEFINITIONS

$\boldsymbol{\theta = (w, b)}$	Set of network parameters (weights and biases),
$\boldsymbol{I}$	Set of training inputs,
$\boldsymbol{T}$	Set of testing inputs,
$\boldsymbol{L(\theta, I)}$	BCE loss function,
$\boldsymbol{AUC(\theta, I)}$	Area under the ROC curve,
$\boldsymbol{b}$	Size of input batches,
$\boldsymbol{B_i}$	$i-th$ batch,
$\boldsymbol{e}$	Epoch,
$\boldsymbol{M(\theta, Bi, n)}$	Output of the Memcomputing routine,
$\boldsymbol{n}$	Nb of time steps for the self-organizing circuit simulation,
$\boldsymbol{th}$	Threshold for the update of the learning parameters,
$\bar{r}$	Restart rate for the update,
$\boldsymbol{n_{layer}}$	Number of trainable layers,
$\boldsymbol{s^j_{input}}$	Number of input nodes of the layer j,
$\boldsymbol{\|*\|}$	L^2 norm,
$\boldsymbol{normmax}$	Maximum norm allowed for the weights.