Supplementary Information

Charles Mackin[1], Malte Rasch[2], An Chen[1], Jonathan Timcheck[3], Robert L. Bruce[2], Pritish Narayanan[1], Stefano Ambrogio[1], Manuel Le Gallo[4], S. R. Nandakumar[4], Jose Luquin[1], Alexander Friz[1], Abu Sebastian[4], Hsinyu Tsai[1] & Geoffrey W. Burr[1]

[1]*IBM Research–Almaden, San Jose, CA, USA*

[2]*IBM Research–Yorktown, Yorktown Heights, NY, USA*

[3]*Stanford University, Stanford, CA, USA*

[4]*IBM Research–Zürich, Switzerland*

Hardware-Aware Training Hyper-parameter Scan The following plots show floating-point and hardware-aware training simulations for the two-layer LSTM, ResNet-32, and BERT base networks referenced in this work. Hyper-parameters were scanned to achieve the best possible accuracies each DNN given numerous hardware imperfections. In each case, the hardware-aware trained networks either reach iso-accuracy or are very close. Floating-point reference training simulations, which represent the typical non-hardware-aware trained DNNs, are included as a reference for comparison.

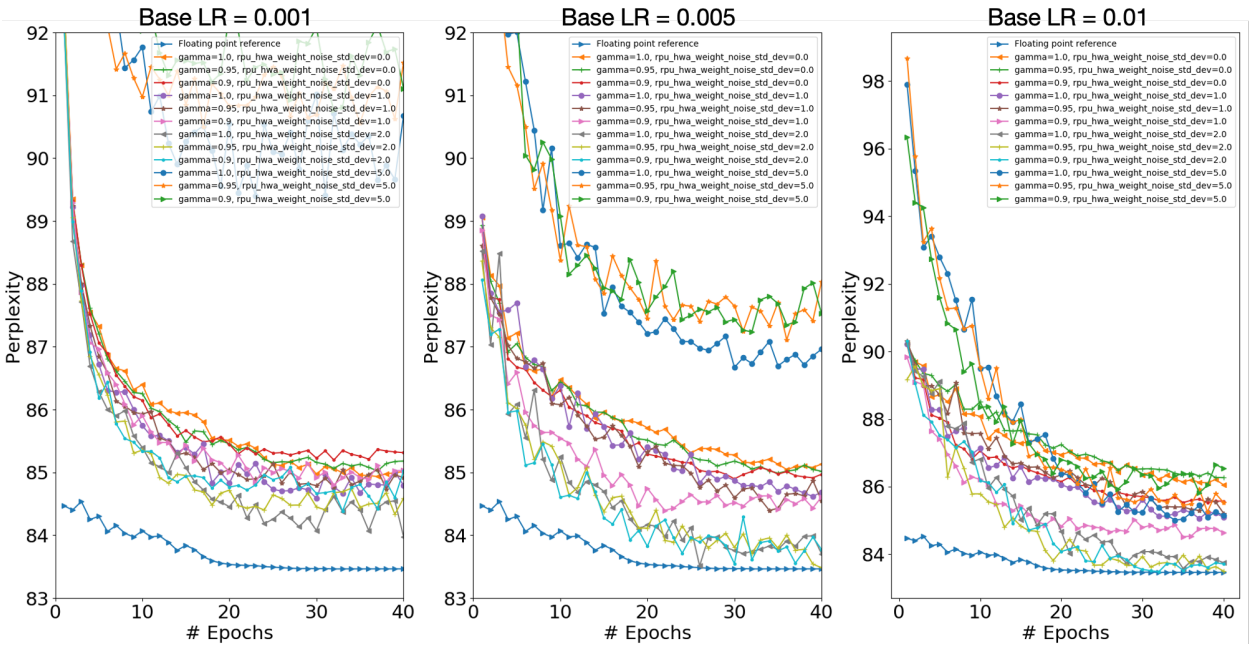

Figure 1: Hyper-parameter scan for the hardware-aware training of the two-layer LSTM network on the Penn Tree Bank dataset. Plots depict some of the hyper-parameters scanned while optimizing the hardware-aware training, which include the base learning rate, learning rate decay factor gamma, and weight noise. A floating point reference training (blue triangles) is included to show that optimized hardware-aware training is able to reach equivalent accuracy.

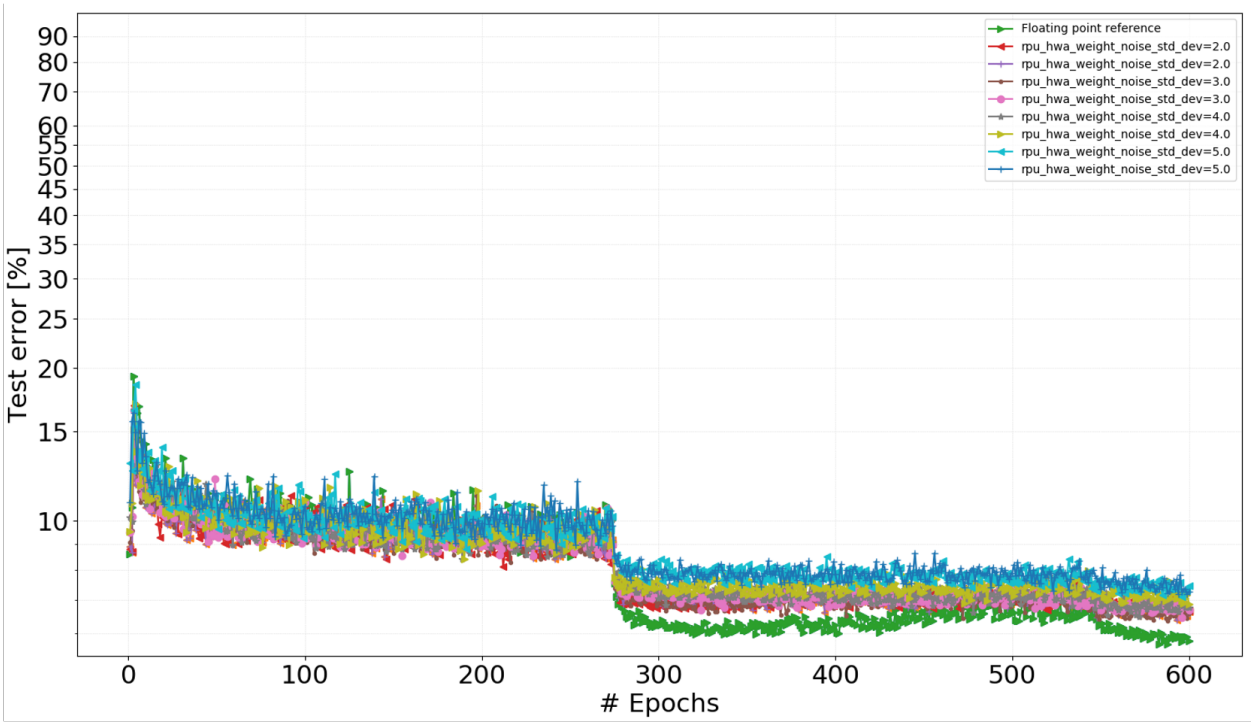

Figure 2: Hyper-parameter scan for the hardware-aware training of the ResNet-32 network on the CIFAR-10 dataset. A floating point reference training (blue triangles) is included to show that optimized hardware-aware training is able to reach equivalent accuracy.

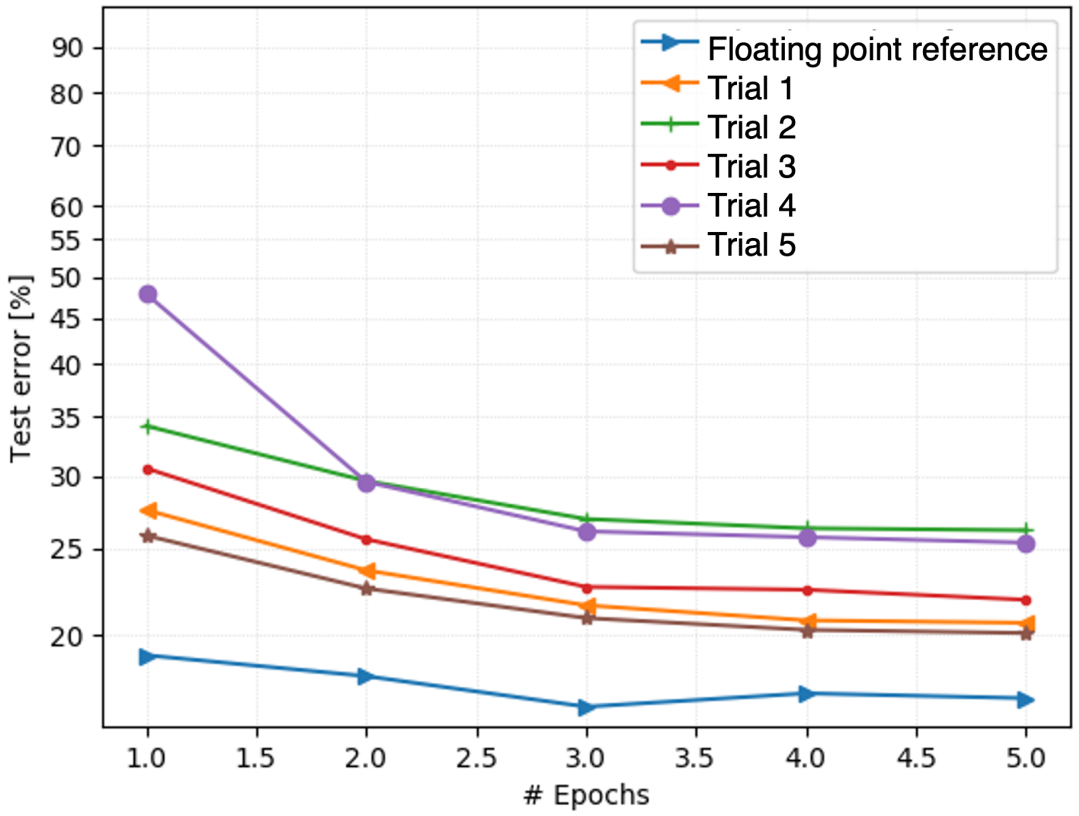

Figure 3: Hardware-aware training of the BERT base on the MNLI dataset. A floating point reference training (blue triangles) is included to show that optimized hardware-aware training is able to reach near-equivalent accuracy.

Hypercube de-normalization details Each hardware synaptic weight is defined by $W = F(G^+ - G^-) + g^+ - g^-$. During optimization, the scale factor F is fixed to some integer value based on the hardware (e.g. 1, 2, 4). Therefore, we have four conductances. To reduce computational expense, we first note that we do not have to explore the entirety of the four-dimensional space. For instance, if we wish to program an analogue weight of 10 μS, the vast majority of potential conductance values $G^+,\ G^-,\ , g^+$ and g^- will not sum to 10 μS. We can limit the exploration space, by realizing that we actually only wish to explore a hyperplane where $W = F(G^+ - G^-) + g^+ - g^-$. In essence, g^- can be defined as $g^- = W - F(G^+ - G^-) - g^+$, which allows us to eliminate a vast majority of conductance combinations that would produce weight values no where near the desired value and limit our exploration to a three dimensional space. We can add some tolerance ΔG to allow some wiggle room around the hyperplane. This allows the weight programming optimization to find solutions that deliberately slightly over-program or under-program certain weights to compensate for drift and achieve overall better inference accuracy over the time interval of interest.

What further complicates the problem, however, is that each G value is bounded by the previously selected G values. In our baseline conductance model, for instance, the PCM conductance range is limited to $0 - 25\mu S$. Now let's say $F = 2$ and we wish to program a weight of 75 μS. If G^+ is 25 μS, then G^- must be zero and g^+ must be at the limit of 25 μS. In this way, we see that there exist inter-dependencies between conductance values. To avoid the complexity of dealing with a long list of dependent constraints (four for each target weight) in a constrained optimization problem, we instead opt to perform the optimization within a hypercube, which simplifies the constraints, and transform those values into an eligible conductance combinations (denormalization)

in an intermediate step. This hypercube denormalization is described by the following equations.

Hypercube Denormalization: $Denormalize(\{x_{0j}, x_{1j}, x_{2j}, x_{3j}\}) \rightarrow \{G_j^+,\ G_j^-,\ g_j^+,\ g_j^-\}$

$$G_j^{+,min} = \text{max}((\beta_{hw} W_j - g_{max} + g_{min})/F + g_{min} - \Delta G, g_{min})$$

$$G_j^{+,max} = \text{min}((\beta_{hw} W_j - g_{min} + g_{max})/F + g_{max} + \Delta G, g_{max})$$

$$G_j^{+} = \text{clip}(x_{0j}(G_j^{+,max} - G_j^{+,min}) + G_j^{+,min}, g_{min}, g_{max})$$

$$G_j^{-,min} = \text{max}(-(\beta_{hw} W_j - g_{min} + g_{max})/F + G_j^{+} - \Delta G, g_{min})$$

$$G_j^{-,max} = \text{min}(-(\beta_{hw} W_j - g_{max} + g_{min})/F + G_j^{+} + \Delta G, g_{max})$$

$$G_j^{-} = \text{clip}(x_{1j}(G_j^{-,max} - G_j^{-,min}) + G_j^{-,min}, g_{min}, g_{max})$$

$$g_j^{+,min} = \text{max}(\beta_{hw} W_j - F(G_j^{+} - G_j^{-}) + g_{min} - \Delta G, g_{min})$$

$$g_j^{+,max} = \text{min}(\beta_{hw} W_j - F(G_j^{+} - G_j^{-}) + g_{max} + \Delta G, g_{max})$$

$$g_j^{+} = \text{clip}(x_{2j}(g_j^{+,max} - g_j^{+,min}) + g_j^{+,min}, g_{min}, g_{max})$$

$$g_j^{-,min} = \text{max}(-\beta_{hw} W_j - F(G_j^{+} - G_j^{-}) + g_j^{+} - \Delta G, g_{min})$$

$$g_j^{-,max} = \text{min}(-\beta_{hw} W_j - F(G_j^{+} - G_j^{-}) + g_j^{+} + \Delta G, g_{max})$$

$$g_j^{-} = \text{clip}(x_{3j}(g_{max}^{-} - g_{min}^{-}) + g_{min}^{-}, g_{min}, g_{max})$$

Weight Programming Optimization Results for Various Other Device Model Combinations

Here we test weight programming optimization on various additional device model combinations to show that it consistently improves inference accuracy for analogue memory-based DNNs. Additional results are provided for the two-layer LSTM evaluated on the Penn Treebank dataset and ResNet-32 evaluated on the CIFAR-10 dataset. Additional results are not provided for BERT base evaluated on the MNLI dataset due to simulation time constraints.

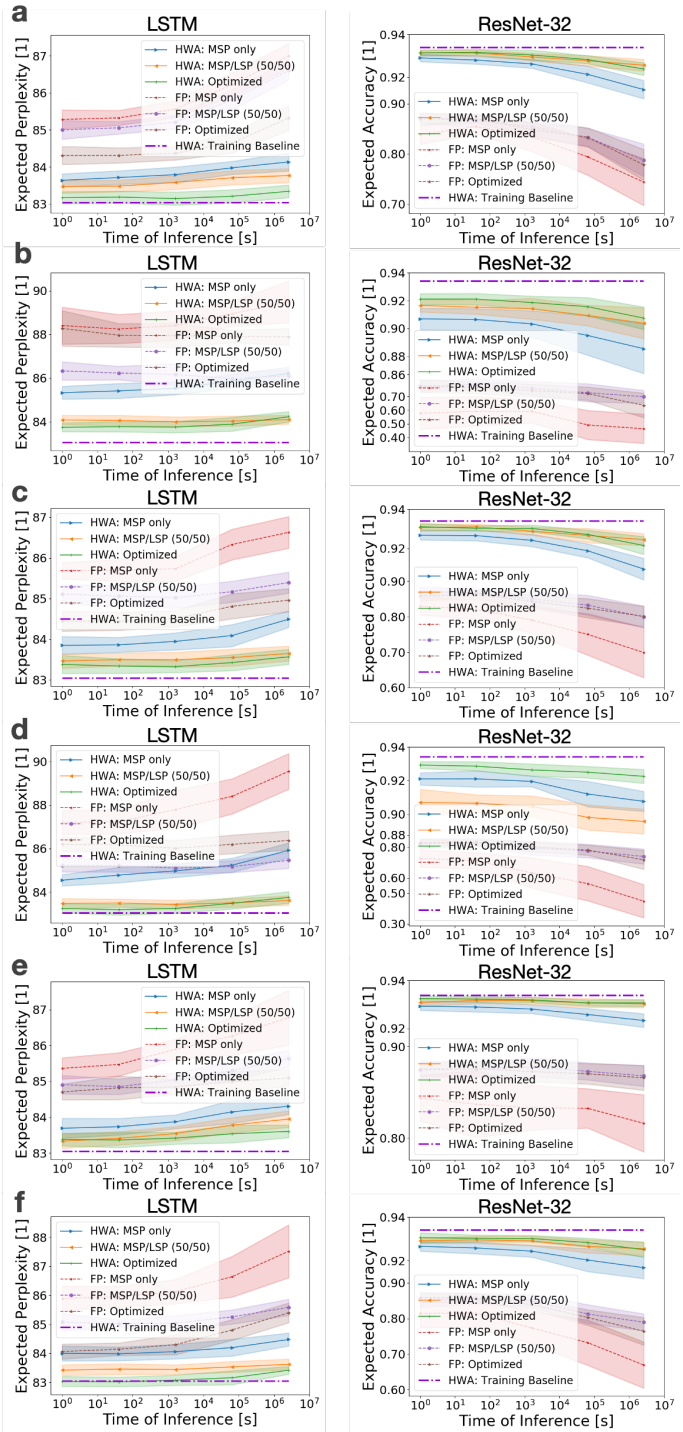

Figure 4: Additional weight programming optimization inference accuracy results for LSTM (Penn Treebank) and ResNet-32 (CIFAR-10) networks using various device model combinations: a) ZeusBaseline, b) Liner A, c) Liner B, d) dGSTa with Liner, e) dGSTb with Liner, f) IRPS with Liner. Results show that weight programming optimization framework consistently enhances inference performance over time and helps DNNs approach iso-accuracy. Simulations were not repeated for BERT base on the MNLI dataset because these take considerably longer to evaluate.