Walking Noise: Understanding Implications of Noisy Computations on Classification Tasks

Hendrik Borras*
Institute of Computer Engineering
Heidelberg University, Germany
hendrik.borras@ziti.uni-heidelberg.de

Bernhard Klein*
Institute of Computer Engineering
Heidelberg University, Germany
bernhard.klein@ziti.uni-heidelberg.de

Holger Fröning
Institute of Computer Engineering
Heidelberg University, Germany
holger.froening@ziti.uni-heidelberg.de

***Abstract*—Machine learning methods like neural networks are extremely successful and popular in a variety of applications, however, they come at substantial computational costs, accompanied by high energy demands. In contrast, hardware capabilities are limited and there is evidence that technology scaling is stuttering, therefore, new approaches to meet the performance demands of increasingly complex model architectures are required. As an unsafe optimization, noisy computations are more energy efficient, and given a fixed power budget also more time efficient. However, any kind of unsafe optimization requires counter measures to ensure functionally correct results.**

This work considers noisy computations in an abstract form, and gears to understand the implications of such noise on the accuracy of neural-network-based classifiers as an exemplary workload. We propose a methodology called "Walking Noise" that allows to assess the robustness of different layers of deep architectures by means of a so-called "midpoint noise level" metric. We then investigate the implications of additive and multiplicative noise for different classification tasks and model architectures, with and without batch normalization. While noisy training significantly increases robustness for both noise types, we observe a clear trend to increase weights and thus increase the signal-to-noise ratio for additive noise injection. For the multiplicative case, we find that some networks, with suitably simple tasks, automatically learn an internal binary representation, hence becoming extremely robust. Overall this work proposes a method to measure the layer-specific robustness and shares first insights on how networks learn to compensate injected noise, and thus, contributes to understand robustness against noisy computations.

***Index Terms*—noisy training, noisy computations, analog computing, robustness, neural networks**

I. Introduction

Over the last two decades the computational demands of modern machine learning solutions have increased continuously at an exponential scale. In the last few years in particular, the speed at which these requirements grow has stepped up significantly [1]. To keep execution times and energy utilization per inference at a manageable level one might hope that advances in digital technology such as new process nodes and improved architectures present a solution. Unfortunately, as shown by Thompson et al. [2], this is not the case. Instead, the growth in compute performance cannot keep up by orders of magnitude. Therefore, new architectures and technologies are coming under close consideration. One particularly interesting approach is to reduce the reliability of computations in order to improve the corresponding energy consumption. Fundamentally, reduced energy consumption allows to perform more computations in a given fixed power budget, thus increasing compute performance.

In this regard, multiple recent works have considered such *noisy hardware* to improve energy efficiency and thus performance, for instance based on analog electrical computations [3–5] and analog optical computations [6, 7]. For completeness, similar applies to emerging memory technologies such as resistive RAM [8]. Considering the example of analog electrical systems, computations can be represented using the physical laws of network analysis, such that for instance the input operands of a Multiply-accumulate (MAC) operation are current pulses, with the pulse's length and amplitude representing activation and weight, respectively. A single multiplication is then the time integral over this pulse, therefore charge. While in theory this analog integration is not consuming energy, in practice effects like inductance, resistance and thermal noise consume small amounts of energy. Nevertheless, analog computations are, for up to 8 bits of precision, considered superior to their digital counterpart in terms of energy efficiency [3].

However, as unreliable computations and memory accesses might obviously have substantial implications on the quality of the overall result of a given application, methods improving resilience against such unreliable hardware are essential. For instance, with *Noisy Machines* Zhou et al. [9] explores the use of knowledge distillation from a digitally trained teacher network to a student network with noise injection. Similarly, training as a counter measure to improve resilience against noise by slowly exposing a neural network architecture to an increasing amount of noise has proven to be effective [10]. In general *noisy training* has been established as vital training methodology for different noisy hardware architectures [11, 12, 9].

This work is part of the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No 886385), and supported by the Austrian Federal Ministries for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) and for Labour and Economy (BMAW), represented by the Austrian Research Promotion Agency (FFG), and the federal states of Styria, Upper Austria and Tyrol. The authors additionally acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

*These authors share first authorship, with different emphasis on methodology, experimentation, data analysis and research narrative.

In the present work, we are concerned with an assessment of the robustness of various model architectures against unreliable hardware. In detail, we propose a method to measure the implications of different types and amounts of injected noise on the prediction accuracy of the model architecture and analyze the inherent defense strategies of neural networks, when trained with noise injection. We apply this method on different architecture and data sets, identify general trends, and discuss early insights from these experiments.

While usually both, additive and multiplicative noise types are found in almost any technology, typically one type is dominant. And it remains an open debate in the community which of these is dominant for a particular analog electrical or optical technology. In particular, it seems that either contribution highly depends on the chosen technology, with examples including [5, 13, 14]. Therefore, we consider both additive and multiplicative noise. Ultimately, this work makes the following overarching contributions:

1) We propose a methodology that assesses the implications of different noise sources on prediction accuracy, which is based on walking selectively through the architecture assessing robustness of the individual layers by means of the "midpoint noise level" metric, thus allowing to understand the sensitivity of different architecture components.
2) We investigate the implications of additive noise, with regard to overall sensitivity for different architectures and their components, and highlight insights from a variety of experiments and observed artifacts. As we will see, one can observe that model architectures trained with additive-noise-injection tend to adapt their parameters to increase robustness and sustain accuracy.
3) Similarly, we apply an identical experimentation to multiplicative noise, again highlighting observations and insights. In this case, we observe that the parameters of the architecture show a substantially different behavior, leading to extreme resilience against noise.
4) For all experiments the influence of batch normalization layers is evaluated. Results suggest that the presence or absence of BatchNorm (BN) layers significantly impacts the way the network learns to deal with injected noise.

The remainder of this work will provide a detailed review of related work, after which it presents the proposed methodology of "Walking Noise". This is followed by a detailed evaluation of additive and multiplicative noise, respectively, in combination with a discussion of observations and insights. Finally, we present a short summary as well as an outlook on future work in this context.

II. Related Work

Since the early days of machine learning, noise injection has been considered as a generalization method [15, 16]. Recently, Jiang *et al.* compared it with other methods like weight decay and early stopping to reduce overparametrization [17].

With the advent of adversarial attacks, the search for defense strategies has become an important research topic. Alongside adversarial training [18], noisy training has proven to be a very effective defense, considering for instance globally injected additive Gaussian noise [19] or ensembles with layer-wise noise injection [20]. Liu *et al.* investigate how the robustness of continuous neural networks [21] against adversarial attacks can be improved through noise injection — differentiating between diffusion (Gaussian distributions) and jump term (dropout) randomness [22].

While most of these related works evaluate input sensitivity and noise tolerance at the network inputs, noisy hardware rather affects computations, thus in particular the inherent dot products of neural networks. To counteract accuracy degradation due to noisy computations, noisy training has also been successfully applied: while [11] and [12] inject additive zero-mean Gaussian noise with different variances during training, *Noisy Machines* [9] extends this further with knowledge distillation. Moreover, the latter was able to explain the higher sensitivity of deeper model architectures in comparison to wider model architectures, as a loss of information through a mutual information analysis.

In particular cases, noisy hardware has been used as an inherent advantage for the deployed neural network. Cappelli *et al.* explicitly increase the robustness against adversarial attacks, by exploiting the noise in an electrical analog circuit [23]. Wu *et al.* used optoelectronic noise for a generative adversarial network with noisy training [13].

While most related works show the benefit of *noisy training* for generalization, increased robustness and as an essential method for working with noisy analog hardware, this work aims to shed light on the internals of model architectures explicitly trained with noise.

III. Methodology

A. Noise injection

Depending on the accelerator, noise effects can appear at varying locations during the inference of a neural network. Considering that most Machine Learning (ML) accelerators are based on a matrix multiplication unit as central computational unit, three places for noise injection become apparent: (1) at the weight readout, (2) inside the calculation, (3) when forwarding activations to subsequent layers. In this work we decided to focus our investigation on the last injection point, since this point can combine effects from the first two injection points.

Furthermore, we consider the additive and multiplicative injection of Gaussian noise, due to its widespread observation in natural processes and success in other works on noise injection [9, 11]. To inject noise without systematic bias, we sample from a Gaussian with zero mean for the additive case and with a mean of one for the multiplicative case. While in physical accelerators one would likely see a mix of both noise types, we assume that usually one type dominates. We thus study exclusively either additive or multiplicative noise.

Such noise is injected either only during inference or during training and inference. Where the first approach corresponds to running an unmodified model on a noisy accelerator, the latter corresponds to hardware-in-the-loop training, which has

been shown to improve network accuracy significantly [10]. While this work shows results for both cases, we primarily focus on effects appearing with noise injection during training, since these more closely reflect how one would use an analog accelerator to achieve the highest possible prediction accuracy.

B. Global and walking noise

To inject noise at the activations of a given neural network, we apply a custom noise module after each layer is executed. With this approach we can selectively inject noise at any given point of the activation path of a network, even directly at the input and output. Similar to Straight-Through Estimators used in Quantization-aware training [24], we only inject noise in the forward path, not the backward path.

For our initial experiments we injected noise globally with the same intensity at all layers of the network. While these experiments already give an initial impression of the sensitivity of a network to noise, no information is gained about the networks internal behavior. To probe how the internal structure of a model architecture reacts to noise injection, we inject noise exclusively at a single layer. By moving the injection point between experiments, the noise is then walking selectively through the network, coining the term **Walking Noise**.

All experiments with different "Walking Noise" injection points are independent of each other, and can thus be trained in parallel. To ensure that we observe all notable effects, we increase the noise until the network converges to the accuracy's lower bound a_{min}, which is in general equivalent to random predictions[1].

C. Data sets and model architectures

We consider two Image Classification (IC) tasks (MNIST [25], CIFAR-10 [26]) and one Natural Language Processing (NLP) task (Google Speech Commands (GSC) v2 [27]). Due to space constraints, we focus on reporting CIFAR-10 results, which transfer to MNIST and GSC if not otherwise stated. Only GSC was preprocessed, by using MFCCs with settings according to the MLPerf Tiny benchmark [28], in particular considering the preprocessing and loss function reweighting as reported by [29].

For this initial work we consider three different model architectures, which are a Multi-layer Perceptron (MLP), LeNet-5 [30], and CNN-S [31]. The MLP consists of three fully-connected (FC) layers with 64 neurons, ReLU activations and optional BatchNorm between these layers, and is used for all tasks. LeNet-5 is used for IC while — due to a significant difference in input shape — for GSC the CNN-S architecture is used without the low-rank linear layer.

D. Experimental setup

In total over $200,000$ experiments were conducted, using SEML [32], an open-source tool to which we also contributed, to generate the appropriate batch scripts. For each training run, the final accuracy is calculated by averaging the last five epochs. Furthermore, multiple training runs were used to obtain confidence intervals (five repetitions for global noise, three for Walking Noise). These intervals were then used in the regression fits.

IV. Analysis of global noise

For all datasets, we train a given network and noise combination until it converges. As general pattern, one can observe how the network accuracy a degrades with increased noise x (referring to the standard deviation of the injected Gaussian noise) until the network stops learning entirely[2]. We then correlate the average final accuracy a of each experiment with the amount of injected noise x, resulting in data points as shown in Figure 1. The standard deviation of the accuracy resulting from multiple experiment repetitions is visualized using error bars, though barely visible in this Figure.

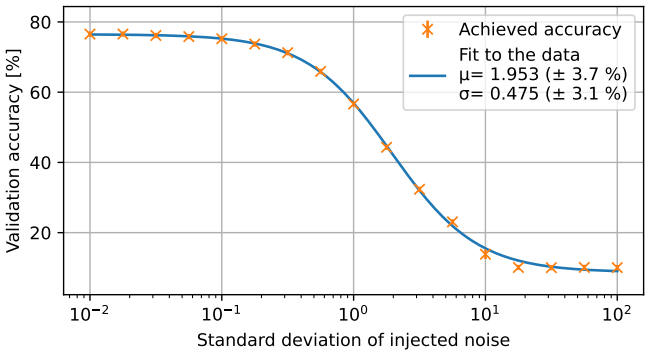


Figure 1. Validation accuracy at inference for LeNet-5/CIFAR-10 with BatchNorm for an increasing amount (standard deviation) of globally injected additive noise. Overlaid is a logistic function as a fit to the data.

A. Quantifying robustness

To quantify the robustness as a key performance metric, we compute the midpoint noise level μ, where the injected noise is such that the network achieves half its overall accuracy, precisely $\delta a = (a_{max} - a_{min})/2$, with a_{max} and a_{min} being the maximal and minimal achieved accuracy, respectively. Equation 1 describes a scaled logistic function which is fitted to the observed data,

$$F(x; \mu, \sigma, \delta a, a_{min}) = \frac{1}{1 + e^{-(x-\mu)/\sigma}} \cdot \delta a + a_{min} \qquad (1)$$

with μ being the location of the midpoint noise level of the curve, σ the curve's slope, and δa and a_{min} being the curves scale and shift factor along the y-axis, respectively. Figure 1 shows an example fit.

Fitting a function to the data additionally gives us the ability to elegantly accommodate data points with uncertainty information. Such uncertainties are a result of fluctuations during training, and usually appear in the middle of the curve while the tails are rather stable. The fit also returns an error estimate, which can be used to assess the reliability of the

[1]Please refer to section VI for a notable exception.

[2]Please refer to Figure 6 in the appendix for an exemplary visualization.

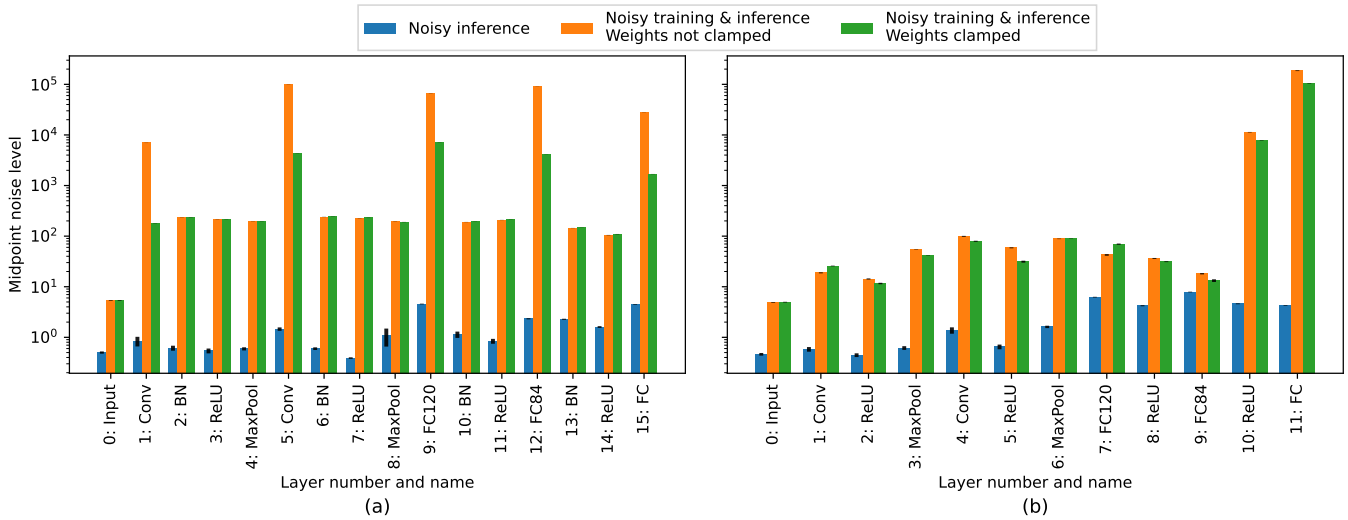


Figure 2. Midpoint noise level for LeNet-5/CIFAR-10 based on Walking Noise, with (a) and without (b) BatchNorm.

obtained result. For the fitting routine we use SciPy's *curve_fit* function, such that the midpoint noise level μ is a result of

$$\mu = \underset{\mu,\sigma,\delta a,a_{min}}{\arg\min} \left\| \frac{F(x;\mu,\sigma,\delta a,a_{min}) - y(x)}{\Delta y(x)} \right\|^2, \forall x \quad (2)$$

where $y(x)$ and $\Delta y(x)$ refer to the observed accuracy and the measured uncertainty for a given noise x, respectively.

B. Robustness results for global noise

Repeating this procedure for all experiment series where noise was injected globally gives a midpoint noise level μ for all datasets, models and regularization methods, as shown in Table I. The BatchNorm did not include additional noise at its layer to keep the amount of overall injected noise constant between experiments. Note that these results are not directly comparable among different noise types and datasets, however, they can indicate how different architectures compare to each other and highlight the influence of BatchNorm layers.

These results provide an early understanding of how robust a model for a particular task is. Still, particular questions, for instance if BatchNorm is helpful or not, remain indecisive. Thus, to understand what happens inside a model architecture, in the following the *Walking Noise* method will be used.

Table I
ROBUSTNESS μ TO GLOBALLY INJECTED GAUSSIAN NOISE.

Dataset	Model	Additive Noise w/o BN	Additive Noise w/t BN	Multiplicative Noise w/o BN	Multiplicative Noise w/t BN
MNIST	MLP	1.69	1.57	0.942	0.952
	LeNet-5	1.34	1.61	0.946	0.966
CIFAR-10	MLP	2.89	2.11	0.655	0.642
	LeNet-5	1.62	1.95	0.625	0.629
GSC	MLP	5.83	2.06	0.670	0.667
	CNN-S	7.89	3.73	0.916	0.772

V. ADDITIVE NOISE

To start out with the *Walking Noise* methodology, the first noise type investigated was additive noise injection. For this noise some rather intuitive expectations are present, since its impact can be understood more easily.

1) Considering constant noise during training, one would expect that a perfect learning procedure should improve its signal-to-noise ratio by learning larger weights, as larger weights will then result in an improved loss under noise — which was also observed by [9].
2) From the vast amount of experience with quantization in the machine learning community one major insight is that particular care has to be taken in choosing the quantization of the input and output layers to avoid excessive loss of accuracy [33]. Since quantization as a lossy compression technique also introduces a kind of noise, these results suggest that hidden layers are more robust to (quantization) noise, which is expected to also hold true for the noise considered here.

To investigate these expectations we combine weight clamping ($w \in [-1, 1]$) with Walking Noise during training. With weight clamping examining the first expectation and walking noise the second. We report the robustness (midpoint noise level μ) as key metric in Figure 2 for training with and without BatchNorm. These plots were created with LeNet-5 on the CIFAR-10 dataset and are representative of what we qualitatively observed for all model architectures and datasets. We report a baseline (blue) without weight clamping and no noise exposed during training, results for noise applied during training (orange), and results for noise during training with clamped weights (green). For more examples, please refer to Figures 10a to 10c, 11a and 11b in the appendix.

A. Impact of batch normalization

Starting with the model architecture with batch normalization, shown in Figure 2(a) it becomes immediately obvious that the network becomes significantly more robust by applying noise also during training, compared to only injecting noise during inference. It is additionally visible that especially layers with learnable parameters (convolutions and fully connected layers) can become much more robust. This is most notable when the model architecture investigated uses batch normalization, but can also be seen to a much smaller extent when the network is used without this regularization technique.

Most of the resistance values of non-learnable layers appear to plateau around a fixed value. We assume that this plateau is directly caused by the BatchNorm layer as it is directly applied after each layer with learnable parameters. Here the BatchNorm effectively nullifies any absolute increase in the activation values, bringing them to an approximate Gaussian distribution with zero mean and unit variance.

However, for layers with learnable parameters the network can introduce significantly larger activation values directly after the layer, which leads to significantly higher robustness. For this type of layer one can observe that while the three hidden layers are most robust, the first and last layers exhibit less robustness, which is in-line with our previously stated second expectation. Furthermore, these layers appear to learn exceedingly larger weights as noise is increased, as highlighted by the lower noise resistance when weight clamping is enabled.

B. Impact of parameter count

The previous observations are much less visible when BatchNorm is disabled, as shown in Figure 2(b). Here one can still clearly see that training with noise injection is more robust than without, however the regularity of the training results, imposed by the batch normalization is gone. Furthermore, one can observe that batch normalization had in fact improved robustness, as the midpoint noise level is now slightly lower for almost all layers.

Excluding layers 10 and 11, one can still see a trend from low robustness at the input to higher robustness at internal layers, back to less robustness at the output. However, the last two layers show exceptional robustness compared to the others. Note that the robustness is exceptionally high for this network, even of the same order of magnitude compared to the network with BatchNorm. We can explain this high resistance in the last layers with to a combination of two effects:

- Compared to previous layers, the overall injected noise is less, since at the output there are only ten elements in the activation tensor, leading to less noisy elements.
- Layer 9 has significantly more parameters (84x120) compared to the number of output activations (10), therefore, it is significantly simpler for this layer to create large activation values, even when using overall small weights. Note that an architecture with BatchNorm cannot make use of this fact, since the last BatchNorm nullifies any absolute increase in the activation values.

C. Impact of weight magnitude

Very surprisingly, disabling BatchNorm results in almost no difference between training with clamped weights and without, which leads to the hypothesis that a model architecture without BatchNorm is less likely to learn arbitrarily large weights. In fact, a weight magnitude analysis can confirm this. For this, we analyzed the weights for a given network closest to midpoint noise level. We then took the mean of the absolute weight values of the affected layer, giving us its average weight magnitude. The average weight magnitudes with and without BatchNorm are then compared, by considering the ratio between clamped and non-clamped weights (Table II).

Table II
WEIGHT MAGNITUDE RATIO BETWEEN CLAMPED AND NON-CLAMPED WEIGHTS FOR LENET-5 ON CIFAR-10.

Layer ID	**1**	**2**	**3**	**4**	**5**
Layer Type	Conv	Conv	FC	FC	FC
with BatchNorm	51	30	13	24	18
without BatchNorm	1.7	0.87	1.1	0.6	1.1

These results appear to reproduce expectations gained from Figure 2 and verify results found by *Noisy Machines* [9], in particular that weights grow tremendously in magnitude without weight clamping when batch normalization is used.

D. Insights

Concluding the results, we observe that exposing noise during training immensely improves accuracy. We also observe that model architectures with BatchNorm show on average higher per-layer resilience both with and without clamped weights. However, when BatchNorm regularization is used it appears that layers with learnable parameters tend to increase the magnitude of their weights drastically.

On a physical device the absolute weight magnitude would likely be clamped by physical limitations or integer quantization, leading to the expectation that weights would likely grow close to the possible maximum value if BatchNorm is enabled.

In general, the second expectation from quantization is reproduced, in that hidden layers are more robust compared to input and output layers. Notable exceptions are the last layers when no BatchNorm layers are used, where the second to last layer appears to be able to significantly increase the absolute value of output activations.

VI. MULTIPLICATIVE NOISE

Initial expectations were that the results for the multiplicatively injected noise would be similar to additively injected noise. This was primarily motivated by the fact that when injecting noise globally the resulting curve shapes between the two injection types were very similar. The only difference in expectation was that the network would not learn larger weights as this would not influence the signal-to-noise ratio in this case.

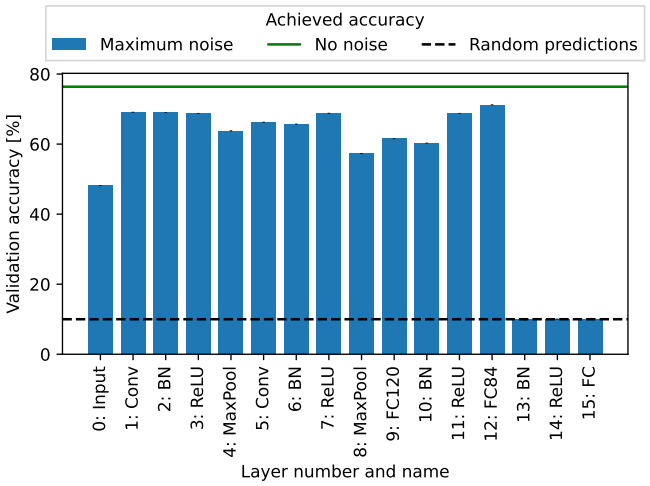


Figure 3. Validation accuracy during inference for LeNet-5/CIFAR-10 with BatchNorm, when injecting noise multiplicatively ($x = 10^{10}$).

A. *Extreme resilience against multiplicative noise*

However, immediately after starting work with the Walking Noise methodology for multiplicative noise it became clear that things were not as cut and dry as expected. In particular when investigating model architectures with BatchNorm regularization, we noticed that some of the layers would not drop to an as low accuracy as previously. Specifically some layers withstood noise injected with a standard deviation up to 10^{10}, without ever dropping by more than 10% points in accuracy on the CIFAR-10 task. Figure 3 shows the drop in accuracy per layer for all LeNet-5 layers on CIFAR-10, when BatchNorm was enabled. These results were extracted in the same fashion as previously, however here we make use of the lower bound a_{min} of the curve fitted to the data (instead of μ, compare eq. 1). This outcome was especially surprising, since we had previously seen that no architecture could withstand injections of larger than 1 on a global scale without significant accuracy degradations (compare Table I). While we did anticipate some increase in noise resistance, seeing the model not drop to random prediction accuracy was not within our expectations.

To better understand what might be enabling this apparently very scalable resistance to multiplicative noise injection, we investigated how the activations of the model behave during inference. To analyze the activations, we extract a checkpoint from a model to which a very high multiplicative noise was applied, and compare it with an equivalent network in which no noise was injected. We then examine the activation distributions for individual layers of these checkpoints using custom hook functions during execution of the model inference. The left column of Figure 4 illustrates the activations for each layer as histograms. The bin height is normalized to the count density, which gives a more natural impression of the bin contents for the log plots, where the space covered by each bin varies. For most of the visualization these histograms are of similar shape, but after the noise injection layer, the activation

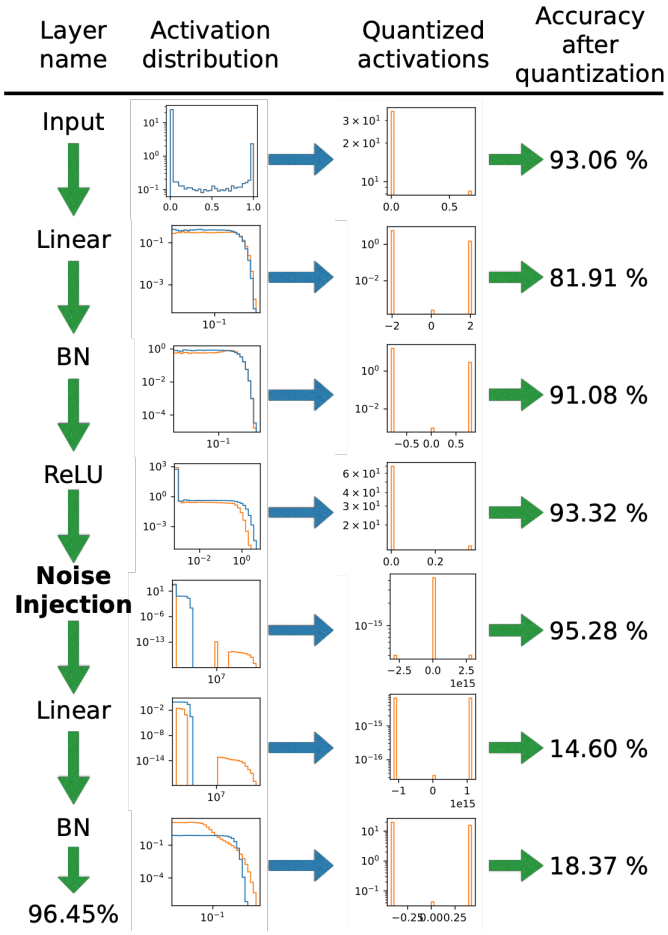


Figure 4. Visualization of self-binarization when injecting multiplicative noise using *Walking Noise* into a MLP on MNIST. The left histograms show the distribution of activation values after the layer is executed: In blue without noise injection; in orange with noise injected at the marked layer. Logarithmic and linear x-axis scales are used to highlight variations between the two networks. On the right side the distribution of activation values with achieved accuracy is shown when applying a simple threshold based quantization to the respective layer.

values split into two peaks of a bimodal distribution. One mode clusters around zero, while the other clusters just below the standard deviation of the injected noise. This suggests that the network is able to self-binarize its activation values into two distinct clusters.

B. *Self-binarization of model activations*

We hypothesize that the network is able to recover information solely from these two peaks, without the need for any exact value within the peaks themselves, since the values within have been effectively randomized by noise injection. The network would then effectively encode a learned binary representation and afterwards decodes it again as a ternary distribution, since the noise acts symmetrically. To investigate if this is what we were observing, we introduced an additional quantization step to alter the in-flight activations during the computation, using a simple threshold-based quantization scheme. To be exact, we selectively quantize a layer after

training into two or three peaks: the values centered around zero were set to be exactly zero and the values around the second/third peak were set to their average. The right side of Figure 4, shows the activation histograms and accuracies after the quantization has been applied to the corresponding layer. Since this operation does not preserve the shape of the peaks, but reduces them to one value, we expect some accuracy degradation. However, the accuracy should drop significantly when the network is not able to tolerate the binarization of the corresponding layer.

Interestingly, the experiments appear to confirm our hypothesis that the network has indeed learned a binary representation by itself. In particular when applying quantization to the output of the noise layer, the accuracy is only marginally impacted, especially when compared to other quantization points.

It thus appears that the distinction between zero and any large number is one way, if not the primary one, in which a model can reliably circumvent multiplicative noise injection. Although this is not as easily visible for other model architectures and datasets, likely due to weight sharing effects for the convolutional models and higher complexity of the other datasets, we still observe that the accuracy for most layers never drops to the equivalent of random predictions.

C. Comparing different datasets and the impact of BatchNorm

Since this form of robustness appeared across all datasets and models we were able to compare how the same model architecture performs on different datasets on a per-layer basis. Figure 5 illustrates the preserved relative accuracy for each layer of the MLP for different datasets. For CIFAR-10 and GSC the maximum injected noise was 10^{10}, while for MNIST it was even higher $\left(10^{16}\right)$ to explore if some accuracy degradation would become visible. The preserved relative accuracy is calculated by taking the ratio of the model accuracy at no noise and the accuracy at maximum noise, thus making it possible to compare different datasets, even if the baseline accuracy is different. It should be noted that accuracy in and of itself is a highly non-linear and network-dependent metric, so while general trends are a helpful insight, the absolute values are to be taken with a grain of salt. In Figure 5 one can see that CIFAR-10 and GSC have a higher complexity compared to MNIST, as the MLP is able to preserve much more accuracy for MNIST. One can also observe the general trend that, as the noise injection happens deeper in the model architecture, the networks ability to counteract the noise becomes better, until the second or third to last layer, when the accuracy sharply drops to random accuracy. We assume that this final drop is due to the network not being able to properly decode any self-learned binarization without a BatchNorm layer behind.

These results do extend to the more complex convolutional model architectures, showing that self-binarization appears to be possible independent of architecture. And in particular more complex model architectures appear to be able to make even better use of this effect when the BatchNorm is enabled, see Figures 8 and 9 in the appendix.

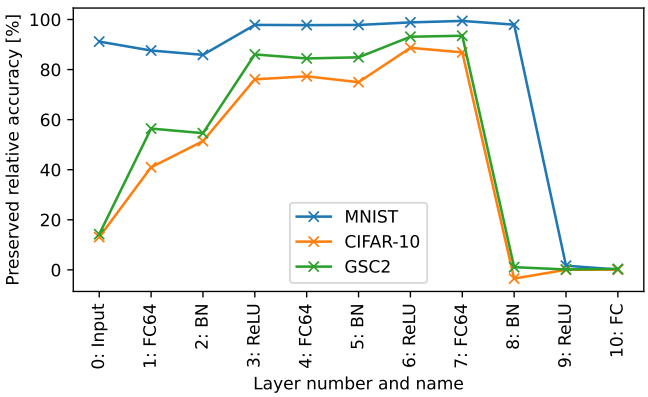


Figure 5. Accuracy preservation for the MLP with BatchNorm with various datasets, injecting noise multiplicatively at a given layer.

This requirement for the BatchNorm to properly handle the self-binarization is strengthened by the observation that without any BatchNorm none of the model architectures were able to make consistent use of this effect. However, there are indications that even without BatchNorm some self-binarization is possible. We observed that the accuracy against noise injection curves (compare Figure 1) would no longer follow a single logistic function, but would first drop to a plateau, before dropping down to random prediction accuracy, as illustrated in Figure 7 in the appendix. This behavior can be described by two stacked logistic functions, suggesting that there are two independent effects at play. We assume that the plateau is caused by intermittent self-binarization, before the network can no longer compensate for the increased activation magnitude and the accuracy drops down completely.

D. Insights

We notice that, similar to additive noise injection, training with multiplicative noise injection increases accuracy results immensely. We further observe that, when BatchNorm is in use, the network is able to self-binarize to make explicit use of the uniqueness of the zero in multiplications and as a path for transmitting information through multiplicative noise. This effect appears to also exist when no BatchNorm is used, however the consistency and overall effectiveness is significantly diminished. When looking at systems, which incorporate both additive and multiplicative noise, we expect that this self-binarization effect is highly dependent on how the two injection types relate to each other. If the additive noise is small enough and not distorting activations too much, then a network may still benefit from exploiting the uniqueness of the zeros and improving accuracy this way.

VII. Summary and Outlook

This work considers the combination of energy-efficient but noisy computations with simple model architectures and datasets to provide an early understanding of the fundamental behavior in terms of robustness. In more detail, we provide an abstract model of noisy computations by adding additive and

multiplicative Gaussian noise to activations between layers. We introduce the metric "midpoint noise level" to assess the robustness of an architecture against noisy computations, and increase the metrics granularity to the different components of a given architecture by employing the "Walking Noise" methodology. Our results include general information on robustness to absolute noise levels, the impact of BatchNorm but also parameter count and weight magnitude. We observed fundamentally different behavior of additive and multiplicative noise. For additive noise we observe increased growth in the magnitude of weight parameters and confirm that components closer to the entry and exit of an architecture are more sensitive to noise. While for multiplicative noise we gain the insight that models with BatchNorms self-binarize during training, which results in extreme noise resistance. In general, this impact of BatchNorm is highly interesting and adds more aspects to ongoing discussions about BatchNorm in general, which is up-to-date still not understood in its entirety [34, 35].

We understand this work as a first step towards a complete understanding of more complex model architectures, such as residual networks and attention-based networks. Ultimately, we are interested in finding the most robust model architecture for a given type of noise and other imperfections such as non-linearities, saturations, and time-/order-dependent behavior of "noisy" computations.

REFERENCES

[1] Dario Amodei and Danny Hernandez, "Ai and compute." [Online]. Available: https://openai.com/blog/ai-and-compute/

[2] N. C. Thompson, K. H. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *CoRR*, vol. abs/2007.05558, 2020. [Online]. Available: https://arxiv.org/abs/2007.05558

[3] B. Murmann, "Mixed-signal computing for deep neural network inference," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, 2021.

[4] G. Cowan, R. Melville, and Y. Tsividis, "A vlsi analog computer/digital computer accelerator," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, 2006.

[5] J. Schemmel, S. Billaudelle, P. Dauer, and J. Weis, *Accelerated Analog Neuromorphic Computing*. Springer International Publishing, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-91741-8_6

[6] X. Lin *et al.*, "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, no. 6406, 2018.

[7] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, vol. 11, no. 7, Jul 2017.

[8] Y. Du *et al.*, "Exploring the impact of random telegraph noise-induced accuracy loss on resistive ram-based deep neural network," *IEEE Transactions on Electron Devices*, vol. 67, no. 8, 2020.

[9] C. Zhou, P. Kadambi, M. Mattina, and P. N. Whatmough, "Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation," *CoRR*, vol. abs/2001.04974, 2020. [Online]. Available: https://arxiv.org/abs/2001.04974

[10] B. Klein *et al.*, "Towards addressing noise and static variations of analog computations using efficient retraining," in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Cham: Springer International Publishing, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-93736-2_32

[11] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 1, 2020.

[12] A. S. Rekhi *et al.*, "Analog/mixed-signal hardware error modeling for deep learning inference," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317770

[13] C. Wu *et al.*, "Harnessing optoelectronic noises in a photonic generative network," *Science Advances*, vol. 8, no. 3, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.abm2956

[14] A. Ash-Saki, M. Alam, and S. Ghosh, "Qure: Qubit re-allocation in noisy intermediate-scale quantum computers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317888

[15] A. Murray and P. Edwards, "Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, 1994.

[16] Y. Grandvalet, S. Canu, and S. Boucheron, "Noise injection: Theoretical prospects," *Neural Computation*, vol. 9, no. 5, 1997.

[17] Y. Jiang, R. M. Zur, L. L. Pesce, and K. Drukker, "A study of the effect of noise injection on the training of artificial neural networks," in *2009 International Joint Conference on Neural Networks*, 2009.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv e-prints*, Dec. 2014.

[19] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00068

[20] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, "Towards robust neural networks via random self-ensemble," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[21] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[22] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh, "How does noise help robustness? Explanation and exploration under the neural sde framework," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.00036

[23] A. Cappelli, R. Ohana, J. Launay, L. Meunier, I. Poli, and F. Krzakala, "Adversarial robustness by design through analog computing and synthetic gradients," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

[24] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[25] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[26] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," 2009.

[27] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[28] C. R. Banbury *et al.*, "Benchmarking tinyml systems: Challenges and direction," 2020.

[29] H. Borras *et al.*, "Open-source FPGA-ML codesign for the MLPerf Tiny Benchmark," in *3rd Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware (MLBench) at 5th Conference on Machine Learning and Systems (MLSys)*, 06 2022.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[31] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontroller," 2017.

[32] D. Zügner, J. Gasteiger, and N. Gao, "SEML: Slurm Experiment Management Library," 2022. [Online]. Available: https://github.com/TUM-DAML/seml

[33] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[34] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[35] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. [Online]. Available: https://doi.org/10.1109/CVPR.2019.00279

APPENDIX

A. Methodology: Additional figures

To extend on how a given network is trained under noise injection, Figure 6 highlights the training process for LeNet-5 on the CIFAR-10 task trained with noise injection. While the network converges after some epochs to non-noise accuracy for small additive noise values, an increasing amount of noise deteriorates the accuracy until it reverts to random prediction accuracy.

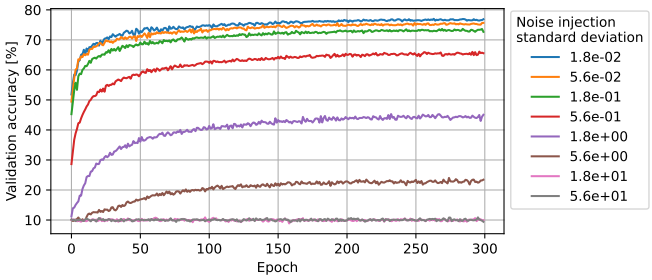


Figure 6. Validation accuracy for LeNet-5/CIFAR-10 with training epochs for globally injected additive noise. One can observe how an increasing amount of noise deteriorates the accuracy until it reverts to random prediction accuracy.

B. Multiplicative noise: Walking Noise without BatchNorm

Extending on the effects of multiplicative noise, Figure 7 shows how accuracy degradation is intermittently paused when employing no Batch Normalization in LeNet-5 on the CIFAR-10 task. These plot in particular show how the data can be described by two stacked logistic functions, highlighting that there are likely two independent effects at play. The example further more demonstrates the usefulness of measuring the uncertainty of datapoints by repeating experiments, as it would otherwise be impossible to reliably fit a function to the parts with large error bars.

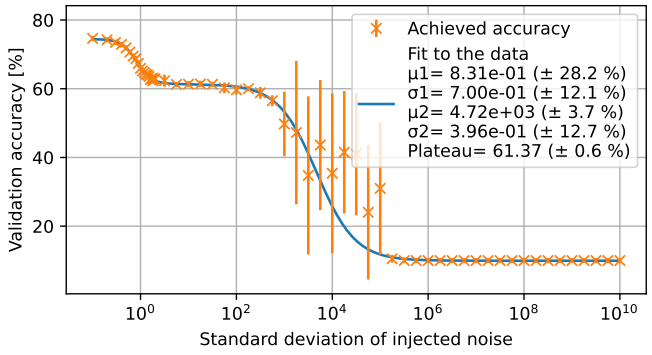


Figure 7. Validation accuracy for LeNet-5/CIFAR-10 with fit of two stacked logistic functions when injecting noise multiplicatively at the third layer in the network without BatchNorm.

C. Multiplicative noise: Preserved accuracy with BatchNorm for other model architectures

To highlight that model accuracy can be preserved for multiplicative noise, given BatchNorm regularization, Figures 8 and 9 show the preserved relative accuracy for the other model architectures investigated in this work. It is particularly notable that the accuracy is preserved to a certain degree for all architectures and datasets, excluding the last three layers.

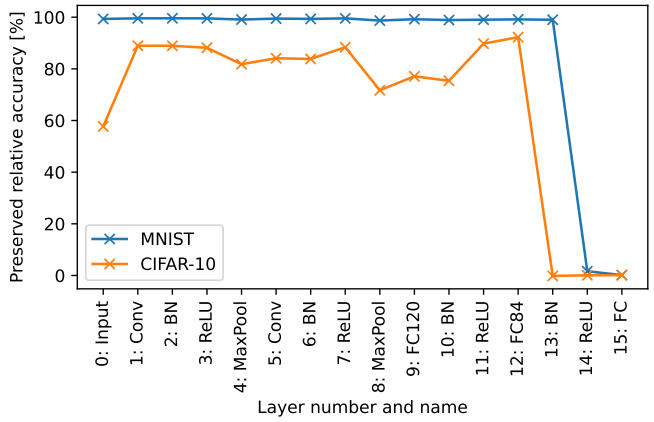


Figure 8. Accuracy preservation for the LeNet-5 during inference on different datasets, when injecting noise multiplicatively at a given point in the network with BatchNorm.

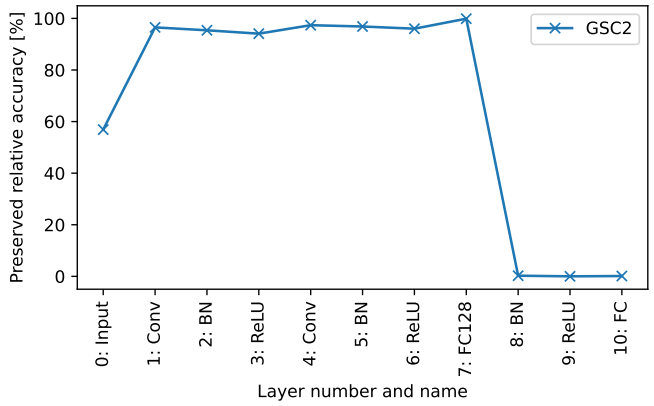
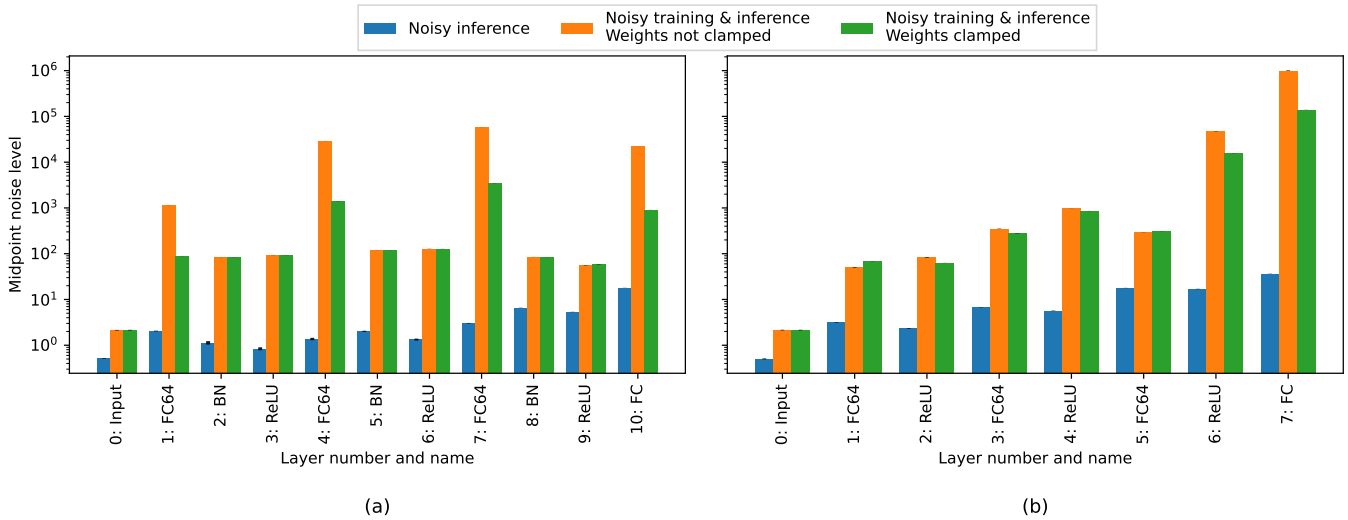


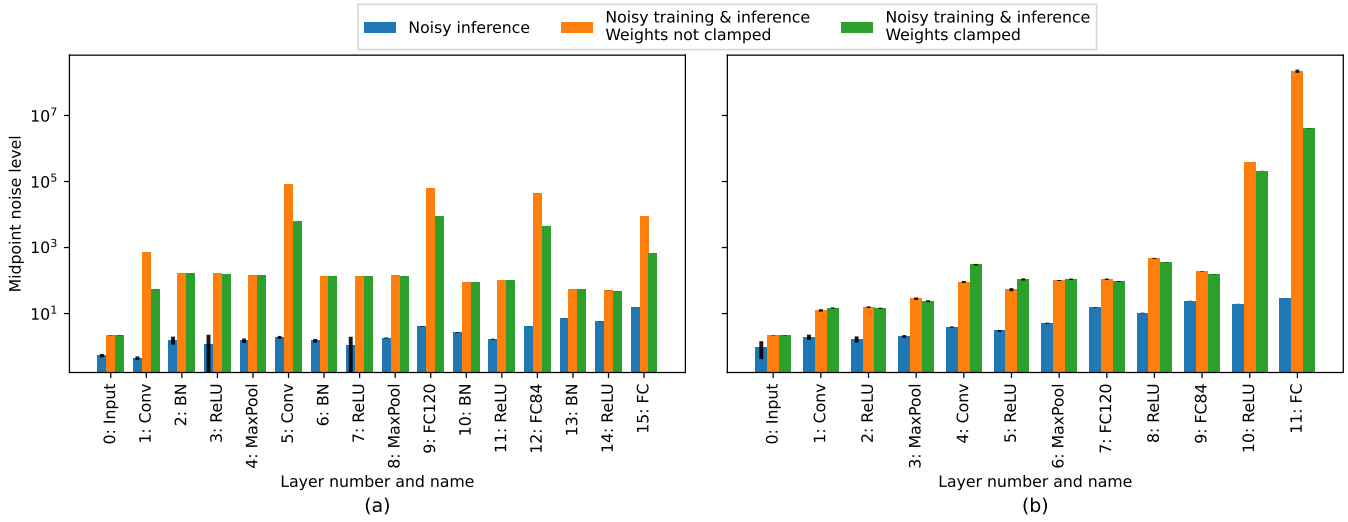
Figure 9. Accuracy preservation for the CNN-S during inference on Google Speach Commands, when injecting noise multiplicatively at a given point in the network with BatchNorm.

D. Additive noise: Other model architectures and datasets

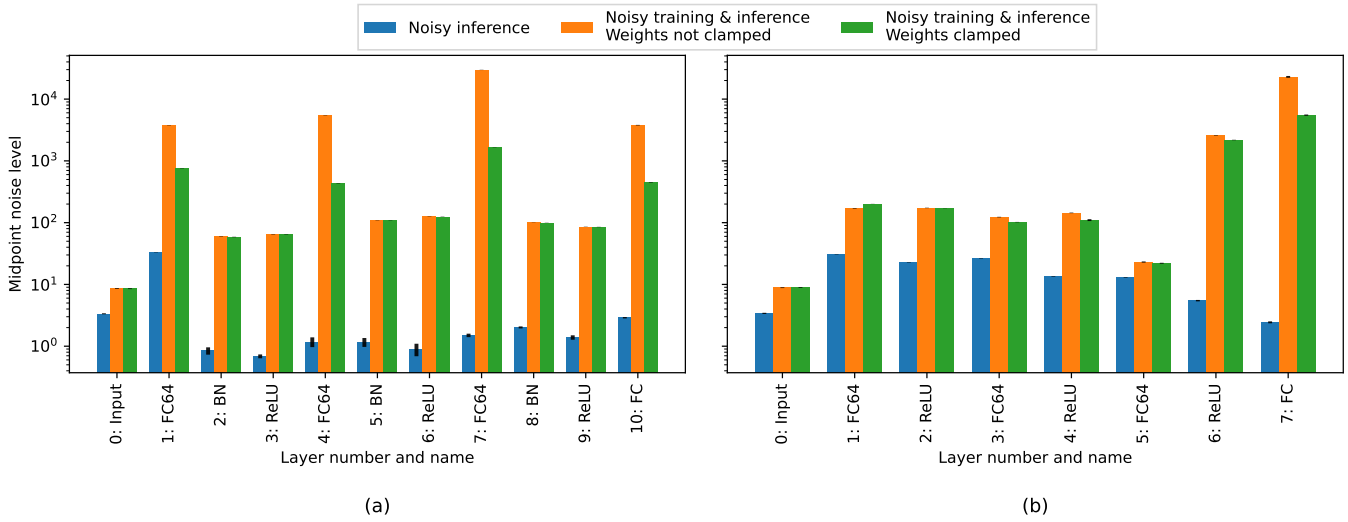
To highlight that the injection of additive noise with the Walking Noise methodology behaves similar on all investigated model architectures and datasets, the following Figures 10a to 10c, 11a and 11b show plots similar to Figure 2. In particular these Figures show results for the MLP and CNN-S architectures and the datasets MNIST, CIFAR-10 and GSC.



(a) MLP on MNIST.

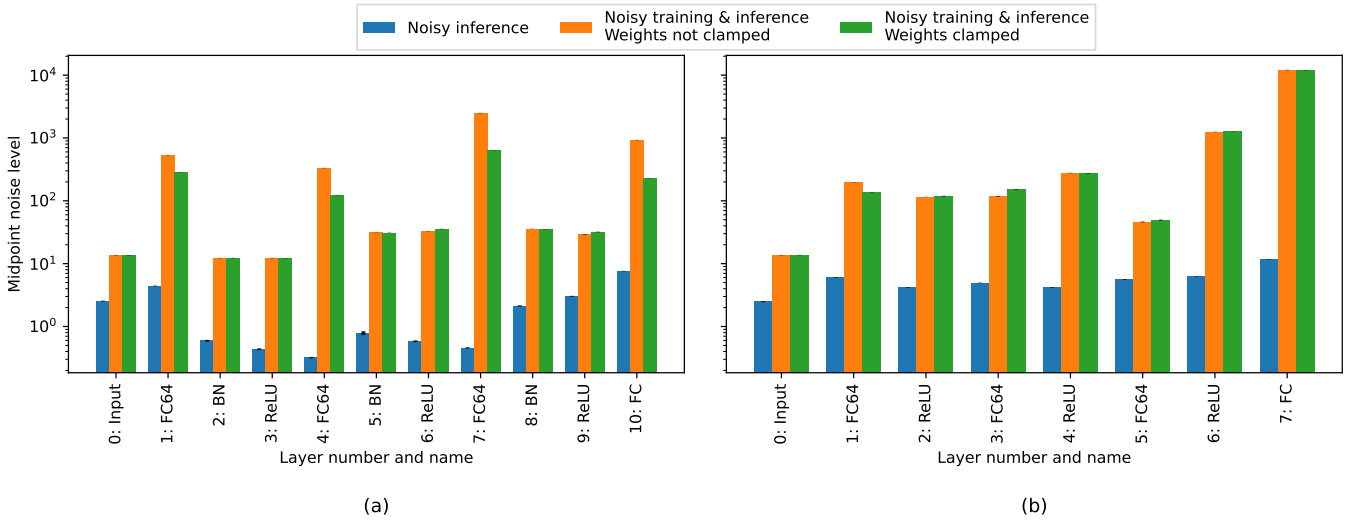


(b) LeNet-5 on MNIST.

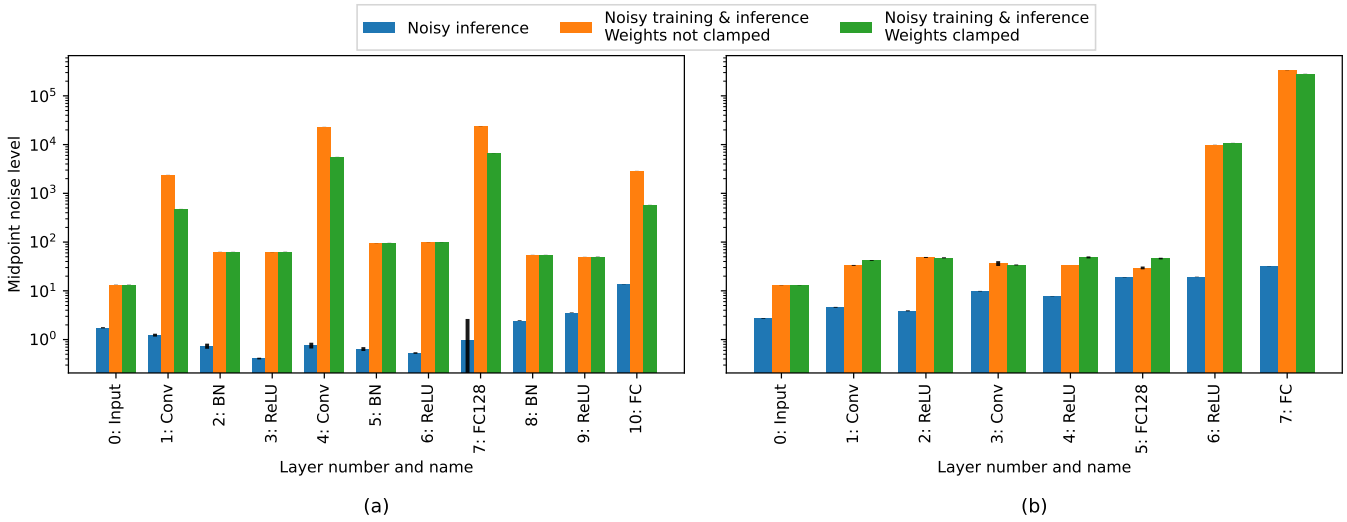


(c) MLP on CIFAR-10.

Figure 10. Midpoint noise level for various model architectures and image classification datasets based on Walking Noise. The x axis shows layer number and name. Figures (a) to the left are with BatchNorm, Figures (b) to the right are without BatchNorm.



(a) MLP on GSC.



(b) CNN-S on GSC.

Figure 11. Midpoint noise level for various model architectures on Google Speach Commands with Walking Noise. The x axis shows layer number and name. Figures (a) to the left are with BatchNorm, Figures (b) to the right are without BatchNorm.