Improving the Training of Neural Networks through Noise Addition

EE338 Digital Signal Processing
Group-25

Shounak Das (21D070068)
Mrudul Nepalchand Jambhulkar (21D070044)
Chinta Siva Madhav (21D070020)

Contents

1 Introduction

Deep neural network architectures offer great performance for tasks in different domains like image processing, speech recognition and natural language processing. Complex architectures like convolutional networks and LSTMs are easier to optimise as compared to the classical feed-forward and recurrent models. These deep networks show great results because of the development of simple and broadly applicable learning techniques like ReLUs[12], dropout[15] , gradient clipping, and weight initialization strategies.
Recently, neural networks have been used in more challenging domains like question answering or program induction, which require more complicated architectures and bring new optimization challenges.
In this project, we will be discussing the effect of the addition of gradient noise in signal processing to optimize the models as it reduces training loss, improves accuracy, and avoids overfitting for the models that train deep neural networks with stochastic gradient descent.

2 Background

Recent work has shown that adding random noise to weights, gradients, or hidden units improves the learning of neural networks.
In methods like weight noise [16] and adaptive weight noise learning [2] is improved by adding random noise to weights. But in these methods, the noise is not annealed and is non-zero at convergence.
The Stochastic Gradient Langevin Dynamics algorithm [20] uses gradients with added noise to accelerate MCMC inference for logistic regression and independent component analysis models. The gradient noise method which is found to be the most effective is very similar to this method.
Different optimization techniques have been used to improve the learning of neural networks, for example, momentum and adaptive learning rates. However, these methods provide good convergence results in convex settings. Adding gradient noise is more suitable for non-convex settings. This method doesn't allow the model to reach local minima too quickly and also allows the early descent to be faster such that the training loss is minimised.

3 Method

The study[14] investigates the efficacy of incorporating time-dependent Gaussian noise into the gradient during the training process. By augmenting the gradient with noise sampled from a Gaussian distribution at each training step t, the technique aims to enhance the training dynamics. Notably, the research indicates that employing annealed Gaussian noise with a decaying variance, as inspired by Welling & Teh [20], yields superior results compared to using fixed Gaussian noise.

The experimental setup involves adopting a schedule for the variance of the Gaussian noise, governed by the formula $\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$. Here, the parameter η is chosen from the set $\{0.01, 0.3, 1.0\}$, and γ is set to 0.55. This strategy results in higher gradient noise levels at the onset of training, gradually diminishing as training progresses.

The review highlights that the introduction of increased gradient noise during the early stages of training serves to push the gradient away from zero, fostering exploration within the optimization process. This approach demonstrates promising potential for improving convergence and enhancing the generalization performance of trained models.

4 Experiments

We will discuss the following experiments :

- Deep Networks (for image classification)
- End-To-End Memory Networks (for question answering) [17]
- Neural Programmers (for question answering) [13]
- Neural Random Access Machines (for algorithm learning) [10]
- Neural GPUs (for algorithm learning) [7]

4.1 Deep Fully-Connected Networks

The impact of noise addition was tested on the neural network for the classification of handwritten digits using the MNIST dataset [11]. Gaussian noise with 0 mean and decaying variance with $\eta = 1$ was used. The network had 20 hidden layers with 50 hidden units. ReLU activation function [12] was used. The weights were initialized from Gaussian with 0 mean and 0.1 standard deviations (simple init). Stochastic gradient descent(SGD) without momentum was used with learning rates 0.1 and 0.01 to train the model .

When trained from simple init and incorporating noise into the gradient, achieved a higher average and best accuracy over 20 runs using each learning rate (total 40 runs).Average is close to 50 percent because small learning rate of 0.01 gives very slow convergence. Adding noise to less dense neural networks (5 layers) did not show much improvement in training.

In the next experiment, gradients were clipped with threshold values of 10 and 100. The noise addition was insensitive to the clipping value. Tuning the threshold can improve the accuracy of the model.

In another experiment, ReLU initialization technique (Good Init) [18, 6] was used, but noise addition did not help much in this case. SGD with carefully tuned initialization, momentum, learning rate, and learning rate decay can offer better accuracy, but such robust initialization is difficult to find in complex deep architectures. In order to test how sensitive these networks are to poor initialization, another experiment was conducted with weights initialized to 0, and the model was known to perform poorly. But, on injection of noise the model reached 94.5 percent accuracy .

Thus, for cases where there is poor initialization, noise addition has proven to be a better technique for optimizing the learning process.

Simple Init with No Gradient Clipping	**Best Acc.**	**Avg. Acc.**
No Noise	89.9	43.1
With Noise	96.7	52.7
No Noise + Dropout	11.3	10.8
Simple Init with Gradient Clipping = 100	**Best Acc.**	**Avg. Acc.**
No Noise	90.0	40.3
With Noise	96.7	52.3
Simple Init with Gradient Clipping = 10	**Best Acc.**	**Avg. Acc.**
No Noise	95.7	51.6
With Noise	97.0	53.6
Good Init [6] with Grad. Clipping = 10	**Best Acc.**	**Avg. Acc.**
No Noise	97.4	91.7
With Noise	97.2	91.7
Bad Init (Zero Init) with Grad. Clipping = 10	**Best Acc.**	**Avg. Acc.**
No Noise	11.4	10.1
With Noise	94.5	49.7

Table 1: Average and best test accuracy percentages on MNIST over 40 runs

The review paper [14] only presented results on the MNIST dataset [11]. Based on the approach discussed, we tested it on the CIFAR10 [9] and ImageNet [3] datasets using the ResNet-34 [5] (has 33 convolutional layers) and ResNet-50 [5] (has 49 convolutional layers) models respectively. Our observations revealed that adding noise indeed enhances performance, and its combination with dropout yields the best results.

Noise level	$\eta = 0.02$	$\eta = 0.04$	$\eta = 0.06$
No noise	72.975%	64.382	46.284%
Gaussian noise injection	73.513%	70.142%	65.285%
Dropout and noise injection	**74.005%**	**71.442**	**67.525%**

Table 2: ResNet-50 [5] on ImageNet at different noise levels: Our method of combining Dropout and noise injection consistently achieves the best accuracy.

Noise level	$\eta = 0.05$
No noise	93%
Gaussian noise injection	93.18%
Dropout and noise injection	**93.56%**
Noise level	$\eta = 0.1$
No noise	90.46%
Gaussian noise injection	91.87%
Dropout and noise injection	**92.83%**

Table 3: ResNet-34 [5] on CIFAR10: Our method of combining Dropout and noise injection consistently achieves the best accuracy.

4.2 End-To-End Memory Networks

Memory networks [17] are used for Question and Answer problems. The model can predict the answer when it is provided with the context and question. The model has an attention mechanism that focuses on the contexts that help best in predicting the answer. In End-to-End networks, a latent attention mechanism was implemented.
A Two-stage training procedure was used :

- Training the networks with linear attention
- Using the weights to warmstart the model with softmax attention

In the experiment with memory networks, gaussian noise with 0 mean and decaying variance and $\eta = 1$ was added after gradient clipping. Fixed standard deviation also shows good results and works best when tuned to 0.001. The number of training epochs was set to 200 to understand the behavior of networks near convergence. This method is tested using a two-stage training procedure. An additional stage is implemented where the network is trained without warmstarting but with softmax attention.
Results showed that warmstarting helps improve learning, and the training and validation errors decreased after adding noise. Adding the noise was more effective for training without warmstarting.

Setting	**No Noise**	**With Noise**
One-stage training (Training error)	10.5%	9.6%
One-stage training (Validation error)	19.5%	16.6%
Two-stage training (Training error)	6.2%	5.9%
Two-stage training (Validation error)	10.9%	10.8%

Table 4: The effects of adding gradient noise to End-To-End Memory Networks. Lower values are better.

4.3 Neural Programmers

The Neural Programmer is a neural network architecture augmented with a small set of built-in arithmetic and logic operations that learn to induce latent programs. It is proposed for the task of question answering from tables [13]. Examples of operations on a table include the sum of a set of numbers or the list of numbers greater than a particular value. Key to the Neural Programmer is the use of "soft selection" to assign a probability distribution over the list of operations. This probability distribution weighs the result of each operation, and the cost function compares this weighted result to the ground truth. This soft selection, inspired by the soft attention mechanism of Bahdanau et al. [1], allows for the full differentiability of the model. Running the model for several steps of selection enable the model to induce a complex program by chaining the operations, one after the other.

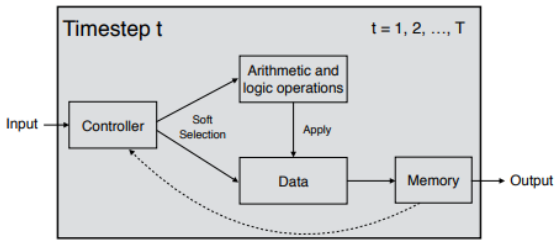

Figure 1: Architecture of the Neural Programmer

In a synthetic table comprehension task, the Neural Programmer takes a question and a table (or database) as input, and the goal is to predict the correct answer. To solve this task, the model has to induce a program and execute it on the table. A major challenge is that the supervision signal is in the form of the correct answer rather than the program itself. The model runs for a fixed number of steps and, at each step, selects a data segment and an operation to apply to the selected data segment. Soft selection is performed at training time so that the model is differentiable, while at test time, hard selection is employed.

Setting	No Noise	With Noise	Dropout	Dropout With Noise
Five columns	95.3%	98.7%	97.4%	99.2%
Text entries	97.6%	98.8%	99.1%	97.3%

Table 5: Effect of adding random noise to the gradient on Neural Programmer. Gradient noise consistently enhances model performance. Yet, the efficacy of combining dropout and noise varies for complex tasks like those with multiple columns or text

Similar to the experiments with Memory Networks, in their experiments with the Neural Programmer, they add noise sampled from a Gaussian distribution with mean 0 and decaying variance. The model is optimized with Adam [8], which combines momentum and adaptive learning rates.

For their first experiment, they train the Neural Programmer to answer questions involving a single column of numbers. They use 72 different hyper-parameter configurations with and without adding annealed random noise to the gradients. They also run each of these experiments for 3 different random initializations of the model parameters and find that only 1/216 runs achieve 100% test accuracy without adding noise. In contrast, 9/216 runs achieve 100% accuracy when random noise is added. The 9 successful runs consisted of models initialized with all three random seeds, demonstrating robustness to initialization. They find that when using dropout [15], none of the 216 runs give 100% accuracy.

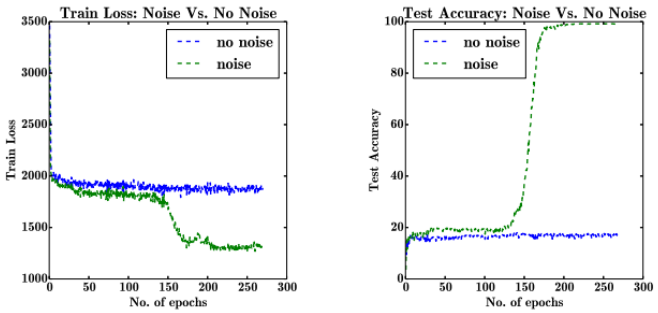

Figure 2: Effect of adding random noise to the gradients in our experiment with 5 columns.

They consider a more difficult question-answering task where tables have up to five columns containing numbers. They also experiment with a task containing one column of numbers and another column of text entries. In all cases, they see that added gradient noise improves the performance of the Neural Programmer. When combined with or used instead of dropout, its performance is mixed depending on the problem, but the positive results indicate that it is worth attempting on a case-by-case basis.

4.4 Neural Random Access Machines

Neural Random-Access Machines (NRAM), [10] are models capable of algorithm learning, enabling data storage, pointer manipulation, and dereferencing. This model comprises a neural network controller, memory, registers, and built-in operations.

The NRAM architecture utilizes an LSTM controller and employs soft attention mechanisms for operation and input selection, rendering the model end-to-end differentiable. To assess its performance, the authors undertake experiments focusing on locating the value of the k-th element in a linked list. Given a pointer to the head of the linked list, the NRAM is tasked with traversing the complex graph to identify the k-th element's value, a nontrivial task due to the random storage of pointers and their associated values.

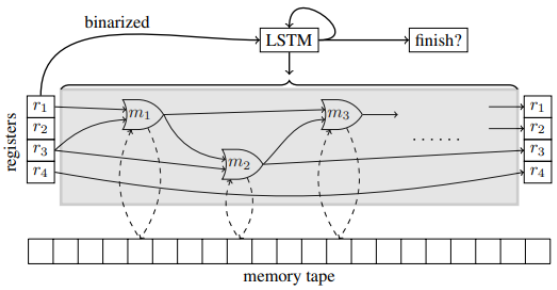

Figure 3: One timestep of the NRAM architecture with R = 4 registers and a memory tape. m1, m2, and m3 are example operations built-in to the model.

Training the NRAM architecture proves challenging, particularly with increased steps and operations, leading to potential instability. To mitigate this, the authors adopt a decaying noise schedule and conduct a thorough hyperparameter grid search, selecting the top three configurations for experimentation. They utilize 100 random initializations for each setting, assessing training accuracy with and without noise.

The experiments reveal the significance of gradient clipping, mainly when training with noise. This observation suggests that excessive gradients can nullify the impact of random noise. Notably, models trained with noise exhibit improved reproducibility rates compared to those trained without noise. While achieving 100% accuracy without noise is feasible, the resulting models prove less robust across multiple random restarts, with significantly fewer initializations yielding correct answers. In conclusion, the findings underscore the effectiveness of incorporating gradient noise in training

Hyperparameter	**No Noise (%)**	**With Noise (%)**
Hyperparameter-1	1	5
Hyperparameter-2	0	22
Hyperparameter-3	3	7
Average	1.3	11.3

Table 6: Percentage of successful runs on k-th element task. Higher values are better

NRAM architectures, particularly in enhancing robustness across multiple random restarts. The authors' comprehensive experimental approach and rigorous analysis contribute valuable insights into optimizing deep network training strategies.

4.5 Convolutional Gated Recurrent Networks

Convolutional Gated Recurrent Networks (CGRNs), also known as Neural GPUs [7], is a novel architecture capable of learning arbitrary algorithms. Unlike traditional sequential architectures like Neural Turing Machines [4], CGRNs leverage parallelism by applying convolutional layers across input data.

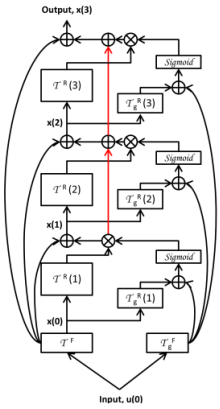

Figure 4: Time-unfolded version of the CGRN with T = 3 (extracted from [19])

CGRNs employ a stack of convolutional layers with tied parameters resembling recurrent networks. Input data, typically symbol sequences, undergoes conversion into a three-dimensional tensor representation, incorporating embedded symbols in the first two dimensions and zero-padding in the third. Modified convolutional kernels, integrating Gated Recurrent Units (GRUs), are applied iteratively across the input sequence. This parallel computation, facilitated by convolution coupled with the gating mechanism, enhances gradient flow and computation efficiency. The additional tensor dimension functions as a working memory, facilitating repeated operations. The final layer outputs the predicted answer.

In the experimental evaluation, the authors employ Neural GPUs for binary multiplication. Training involves 20-digit binary numbers, while testing evaluates the model's ability to multiply 200-digit numbers. Gradient noise, sampled from a Gaussian distribution with mean 0 and decaying variance, is added to the gradient after clipping, following a predefined schedule. The Adam optimizer [8] is utilized for model optimization. The experimental results highlight the efficacy of

Setting	**Error $<$ 1%**	**Error $<$ 2%**	**Error $<$ 3%**	**Error $<$ 5%**
No Noise	28	90	172	387
With Noise	58	159	282	570

Table 7: Number of successful runs on 7290 random trials. Higher values are better. The models are trained on length 20 and tested on length 200

incorporating gradient noise in training CGRNs. Models trained with gradient noise demonstrate increased robustness across various random initializations and parameter settings. Notably, adding gradient noise not only enhances performance, with a significant increase in the number of models achieving <1% error but also improves the overall training robustness. This observation underscores the effectiveness of the simple yet powerful technique of adding gradient noise, particularly in scenarios where extensive random restarts are feasible.

5 Applications

- This technique can be applied to other complex neural network architectures as well like transformers, BERT (Bidirectional Encoder Representations from Transformers), ResNet (Residual Neural Network), VGG (Visual Geometry Group) Net, and Graph Convolutional Networks (GCNs).
- Adding gradient noise to the models can show great results in domains like natural language and image processing. With the ongoing fast developments with AI, there are very complex models that generate text, images, or even videos with some given prompt. These have hundreds of thousands of layers in between. The addition of noise is suitable for some layers, depending on the functionality of the layer.
- By introducing carefully crafted noise during signal processing tasks, such as denoising or filtering, it can help the model learn to distinguish between true signal features and unwanted noise components more effectively.

- In scenarios where signal data contains sensitive information, such as biomedical signals or communication signals, adding noise during processing can help preserve privacy by masking certain features while still allowing for useful analysis or processing.
- Noise also contains some information. In the case of acoustic data, noise may contain info about the environment or the surroundings. Music generation using prompts is at work, and this method of noise addition is beneficial in that case.
- It can also be used for protection for hiding the model or the input data. Financial institutions and the Military use very sensitive information. Their systems use some ML models, too. So, this noise addition helps hide this data.

6 Future Work

- **Optimal Noise Levels:** Investigate optimal strategies for determining the amplitude and distribution of gradient noise for different network architectures and datasets. This could involve adaptive methods that dynamically adjust the noise level during training.
- **Theoretical Analysis:** Further theoretical analysis to understand the effects of gradient noise on the convergence properties of optimization algorithms for deep networks. This could involve proving convergence guarantees or characterizing the behavior of noisy optimization landscapes.
- **Combining with Other Regularization Techniques:** Explore the synergy between gradient noise and other regularization techniques, such as dropout, weight decay, or batch normalization. Investigate how combining these methods can further improve generalization performance and training stability.
- **Transfer Learning and Domain Adaptation:** Investigate how gradient noise can be utilized for transfer learning and domain adaptation tasks. Explore whether adding noise to gradients can help transfer knowledge from pre-trained models to new domains with limited labeled data.
- **Robustness to Adversarial Attacks:** Study the impact of gradient noise on improving the robustness of deep networks against adversarial attacks. Investigate whether adding noise to gradients can make models more resistant to adversarial perturbations without sacrificing accuracy on clean data.

7 Conclusions

The experiments on Deep Fully-Connected Networks, End-To-End Memory Networks, Neural Programmers, Neural Random Access Machines, and Convolutional Gated Recurrent Networks showed that adding noise to the gradient improved the learning of the model, reduced training and validation errors and also helped to avoid overfitting.
This method is helpful if the input data is unseen. The noise toleration of the model also depends on the inputs themselves. Random addition of noise to models makes the model worse off. The addition of noise might work better with data that is somewhat varied. Take the CIFR data we worked with, for example. The objects in the image might have different shapes, orientations, etc. So, regularisation ensures the randomness, and adding noise helps for some models, depending on the input data.
Neural networks with complex architectures where we can't find a robust initialisation easily, this technique of adding noise is very effective for optimisation.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2014.

[2] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning (ICML)*, 2015.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[4] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[7] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Alex Krizhevsky, Vinod Nair, and Geoffrey E Hinton. The cifar-10 dataset, 2009.

[10] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random access machine. *arXiv preprint arXiv:1511.06392*, 2015.

[11] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[12] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[13] Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

[14] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks, 2015.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Mark Steijvers. A recurrent network that performs a context-sensitive prediction task. In *Proceedings of the Annual Conference of the Cognitive Science Society (CogSci)*, 1996.

[17] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[18] David Sussillo and L. F. Abbott. Random walks: Training very deep nonlinear feed-forward networks with smart initialization. *arXiv preprint arXiv:1412.6558*, 2014.

[19] J. Wang and X. Hu. Convolutional neural networks with gated recurrent connections. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 44(07):3421–3435, 2022.

[20] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning (ICML)*, 2011.