ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 17, Issue, 06, pp.33484-33490, June, 2025
DOI: https://doi.org/10.24941/ijcr.49113.06.2025

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

KEY QUESTIONS FOR THE ARTIFICIAL NEURAL NETWORK LEARNER TO ASK THEMSELVES

***Basir Ahamed Khan**

Department of Physics, Murshidabad University, Berhampore, Murshidabad 742101, India

ARTICLE INFO

Article History:
Received 09th March, 2025
Received in revised form
21st April, 2025
Accepted 19th May, 2025
Published online 30th June, 2025

Key words:

Machine Learning, Neural network, Training, Validation loss, Overfitting.

**Corresponding author:*
Basir Ahamed Khan

ABSTRACT

Over the recent years, Artificial Neural Networks (ANNs) have significantly matured, yet numerous unresolved issues persist. Instead of reviewing these open questions, this perspective piece will present my viewpoint on how to tackle problems within the field of ANNs. Specifically, I will outline a set of critical questions we should consider when developing ANN algorithms for machine learning. These questions will center on the fundamental definition of ANNs and their connection to the human brain, the processes of forward and backward propagation, the concepts of training, optimization, and iteration, the methods for minimizing the discrepancy between predicted and actual values, and the recent advancements in model architecture and design.

Citation: Basir Ahamed Khan. 2025. "Key Questions for the Artificial Neural Network Learner to Ask Themselves". *International Journal of Current Research, 17, (06), 33484-33490.*

INTRODUCTION

Artificial Neural Networks (ANNs) have emerged as one of the most powerful tools in modern machine learning (Benjamin *et al.* 2018; D'Amour *et al.* 2022; Sarkar *et al.* 2023), driving advances across domains ranging from computer vision (Loce *et al.* 2013; Sivaraman and Trivedi 2013; Khan *et al.* 2022; Chib and Singh 2024) and natural language processing to robotics and biomedical engineering (Broad *et al.* 2017; Hu *et al.* 2017, Prasad *et al.* 2021; Zhou *et al.* 2022). As their applications continue to expand, so too does the need for learners—whether students, researchers, or practitioners—to develop not just technical proficiency but also a reflective and strategic approach to learning and applying neural network techniques. While there is no shortage of tutorials, courses, and frameworks to help learners build and train neural networks, the true challenge often lies in developing the critical thinking necessary to navigate the complexities of real-world modelling. Understanding the mathematical foundations, interpreting model behaviour, diagnosing performance issues, and making informed design choices all require more than rote learning—they demand a thoughtful inquiry process. This paper aims to support ANN learners by proposing a structured set of key questions they should ask themselves at various stages of their learning and development process. These questions are designed to prompt deeper understanding, guide problem-solving, and foster an iterative mindset conducive to mastery. By consistently engaging with these questions, learners can develop a more robust intuition for neural networks and improve both their theoretical understanding and practical outcomes.

Fundamentals of Neural Network

What is an artificial neural network and how is it inspired by the human brain?: An artificial neural network (ANN) is a computational model designed to mimic the way the human brain processes information (Prieto *et al.* 2016; Gerven and Bohte 2017). It consists of interconnected nodes, or "neurons," which are organized in layers—typically including an input layer, one or more hidden layers, and an output layer. Each connection between neurons has a weight that adjusts as learning progresses, similar to how synaptic strengths in the brain change with experience. The ANN processes data by passing inputs through these layers, using mathematical functions to simulate the firing of neurons and enabling the system to learn patterns, make predictions, or classify data. This structure and learning process are inspired by the biological neural networks in the brain, where neurons communicate through electrical impulses to process and transmit information.

What are the key components of a neural network (e.g., neurons, weights, biases, activation functions)?: The key components of a neural network include neurons, weights, biases, and activation functions. Neurons, also known as nodes, are the basic units that receive inputs, process them, and pass the results to the next layer. Each connection between neurons has an associated weight, which determines the importance of the input signal—higher weights mean stronger influence. Biases are additional parameters added to the input of each neuron, allowing the model to shift the activation function and improve flexibility. Activation functions introduce non-linearity into the network, enabling it to learn and model complex patterns; common examples include the sigmoid, ReLU (Rectified Linear Unit), and tanh functions (Rasamoelina *et al.* 2020). Together, these components work in layers to process data, adjust based on errors during training, and ultimately make accurate predictions or decisions. Fig. 1 shows the schematic diagram of a neural network with one hidden layer.

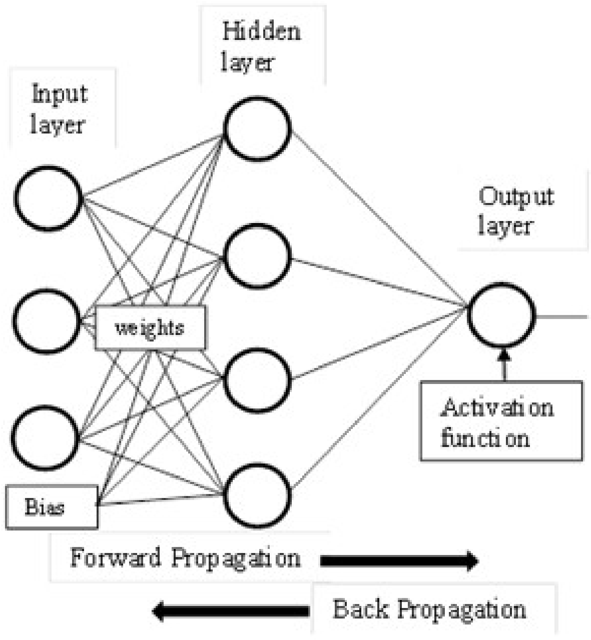

Figure 1. A schematic diagram of Neural Network with one hidden layer

What is the difference between a perceptron and a multilayer perceptron (MLP)?: A perceptron is the simplest type of neural network, consisting of a single layer of neurons that can perform basic binary classification tasks. It takes multiple inputs, applies weights and a bias, and passes the result through an activation function to produce an output. However, perceptrons are limited in their ability to solve complex problems, especially those that are not linearly separable. In contrast, a multilayer perceptron (MLP) is a more advanced architecture that includes one or more hidden layers between the input and output layers. These hidden layers allow the MLP to learn and represent more complex patterns and relationships in the data. Each layer in an MLP uses activation functions to introduce non-linearity, enabling the network to solve problems like image recognition, speech processing, and more, which a single-layer perceptron cannot handle.

Why do we need non-linear activation functions?: We need non-linear activation functions in neural networks because they enable the model to learn and represent complex patterns and relationships in data (Sharma *et al.* 2017, Varshney and Singh 2021). Without non-linearity, a neural network composed of only linear operations—such as weighted sums—would essentially behave like a single-layer model, regardless of how many layers it has, and could only solve simple, linearly separable problems. Non-linear activation functions like ReLU, sigmoid, or tanh allow the network to stack layers in a meaningful way, creating a deep architecture that can approximate complex functions. This capability is essential for tasks such as image classification, language translation, and speech recognition, where the input-output relationships are highly intricate and non-linear (Dubey *et al.* 2022).

Forward and Backpropagation

How does forward propagation work in a neural network?

Forward propagation in a neural network is the process by which input data is passed through the network to generate an output or prediction. It begins at the input layer, where each input value is multiplied by its corresponding weight, and a bias is added. The result is then passed through an activation function to produce the neuron's output. This output becomes the input for the next layer, and the process repeats through all hidden layers until it reaches the output layer. At each layer, the network transforms the data using its current set of weights, biases, and activation functions, gradually building a more abstract representation of the input. The final output is used to make a prediction or classification, which is later compared to the actual result during training to calculate the error for learning.

What is backpropagation and how does it update weights?: Backpropagation is a training algorithm used in neural networks to minimize the error between the predicted output and the actual target value by updating the network's weights. It works by first performing forward propagation to calculate the output and then computing the error using a loss function. This error is then propagated backward through the network, layer by layer, using the chain rule of calculus to determine how much each weight contributed to the error (Zhou 2021). The gradients of the loss with respect to each weight are calculated, and these gradients are used to adjust the weights in the opposite direction of the error—typically with an optimization algorithm like gradient descent. This process allows the network to learn by gradually reducing the error over many iterations, improving its accuracy in making predicions.

What role does the loss/cost function play in training?: The loss, or cost, function plays a critical role in training a neural network by measuring how far the network's predicted outputs are from the actual target values (Crone 2002; Jafarian *et al.* 2018). It serves as a guide for learning, providing a numerical value that represents the model's error for a given set of weights and biases. During training, the goal is to minimize this loss, meaning the network is making more accurate predictions. The loss function is used during backpropagation to compute the gradients, which indicate the direction and magnitude of adjustments needed for each weight in the network. Common loss functions include mean squared error for regression tasks and cross-entropy for classification problems (Alkinanai *et al.* 2020; Matel *et al.* 2022). By continuously evaluating and minimizing the loss, the neural network gradually improves its performance on the training data.

How do gradients flow through the network and what are vanishing/exploding gradients?: Gradients flow through a neural network during the back propagation process, where the error from the output layer is propagated backward through the network to update the weights. This involves calculating the partial derivatives of the loss function with respect to each weight using the chain rule, allowing the network to learn by adjusting weights in the direction that minimizes the loss. However, in deep networks, this process can lead to problems like vanishing or exploding gradients. Vanishing gradients occur when the gradients become extremely small as they move backward through each layer, causing the earlier layers to learn very slowly or not at all. Exploding gradients happen when gradients grow excessively large, leading to unstable training and drastic weight updates. Both issues can hinder the learning process, but they can be addressed with techniques such as using ReLU activation functions, gradient clipping, proper weight initialization, or specialized architectures designed to maintain stable gradient flow.

Training and Optimization

What is the purpose of using an optimizer like SGD, Adam, or RMS prop?: The purpose of using an optimizer like SGD (Stochastic Gradient Descent) (Gardner 1984; Amari 1993; Anuraganand 2021), Adam (Kingma and Ba 2014), or RMSprop (Zou *et al.* 2019) in training a neural network is to efficiently update the model's weights in order to minimize the loss function and improve performance. These optimizers determine how the network learns by deciding the direction and size of each weight update based on the calculated gradients. While SGD updates weights using the average gradient from a batch of data, it can be slow and may struggle with complex loss landscapes. Adam and RMSprop are more advanced optimizers that adapt the learning rate for each parameter, speeding up convergence and improving stability. Adam, in particular, combines the advantages of both RMSprop and momentum, making it one of the most widely used optimizers in deep learning. Overall, optimizers are essential for guiding the training process toward better accuracy and faster convergence.

How do learning rate and batch size affect training?: The learning rate and batch size are two crucial hyper parameters that significantly influence the training of a neural network (Smith *et al.* 2017; He *et al.* 2019; Granziol *et al.* 2022). The learning rate determines the size of the steps the optimizer takes when updating the model's weights. If it's too high, the model may overshoot the optimal solution and fail to converge; if it's too low, training can be very slow or get stuck in local minima. The batch size refers to the number of training examples used to calculate the gradient in a single update. Smaller batch sizes lead to more frequent updates and can help the model generalize better but may introduce noise in the training process. Larger batch sizes provide more stable and accurate gradient estimates but require more memory and can sometimes lead to poorer generalization. Balancing both the learning rate and batch size is key to achieving efficient and effective training (Bruno et al. 2021).

What is overfitting, and how can I prevent it?: Overfitting occurs when a neural network learns the training data too well, capturing not only the underlying patterns but also the noise and random fluctuations (Piotrowski and Napiorkowski 2013; Srivastava *et al.* 2014; Ying 2019). As a result, the model performs well on the training set but poorly on unseen data, indicating poor generalization. This typically happens when the model is too complex relative to the amount of data or when training is too lengthy without proper regularization. To prevent overfitting, several techniques can be used (Schaffer 1993; Defernez and Kemsley 1999; Hawkins 2004; Pothuganti 2018; Santos and Papa 2022): regularization methods like L1 or L2 penalties add constraints to the model's weights, dropout randomly disables neurons during training to promote robustness, and early stopping halts training when performance on a validation set starts to degrade. Additionally, using more training data, simplifying the model architecture, or applying data augmentation can also help reduce overfitting and improve the model's ability to generalize.

How do I evaluate if my model is learning properly?: To evaluate if your model is learning properly, you should monitor its performance on both the training and validation datasets throughout the training process. Key indicators include the loss values and accuracy (or other relevant metrics) over time. Ideally, the training loss should decrease steadily, and the validation loss should follow a similar trend without diverging. If both training and validation performance improve, the model is likely learning effectively. However, if the training loss decreases while the validation loss increases, it may be overfitting. Additionally, plotting learning curves, tracking metrics like precision, recall, or F1-score (especially for imbalanced datasets), and using tools like confusion matrices or ROC (Receiver Operating Characteristic) curves can provide deeper insights into the model's behavior. Regular evaluation ensures the model is not just memorizing data but truly learning meaningful patterns.

Model Architecture and Design

How do I decide the number of layers and neurons per layer?: Deciding the number of layers and neurons per layer in a neural network depends on the complexity of the problem and the amount of available data. For simple tasks, such as linear regression or basic classification, a small network with one or two hidden layers and a modest number of neurons may be sufficient. More complex problems, like image or speech recognition, often require deeper networks with many layers to capture intricate patterns. The number of neurons per layer should be large enough to allow the model to learn, but not so large that it leads to overfitting or unnecessary computation. A common strategy is to start with a simple architecture and gradually increase the depth or width based on performance, using techniques like cross-validation to compare results. Additionally, tools such as grid search, random search, or automated architecture search can help tune these parameters. Ultimately, experimentation and validation are key to finding the right balance for your specific task.

What is the impact of depth vs. width in a network?: The impact of depth versus width in a neural network relates to how the model captures and represents patterns in the data. Depth refers to the number of layers, and increasing it allows the network to learn more abstract and hierarchical features, which is especially useful in tasks like image or language processing. Deep networks can break complex functions into simpler ones, layer by layer. Width, on the other hand, refers to the number of neurons in each layer. Wider networks can model more intricate interactions within a single layer, but they may struggle to capture hierarchical relationships without sufficient depth. While both depth and width can increase a network's capacity, very deep networks can suffer from vanishing gradients or be harder to train, while overly wide networks may overfit or become computationally expensive. Striking the right balance between the two—often guided by experimentation and validation—is essential for building an effective model.

How do different activation functions (ReLU, sigmoid, tanh) affect learning?: Different activation functions like ReLU, sigmoid, and tanh significantly affect how a neural network learns by shaping the output of neurons and influencing the flow of gradients during training. ReLU (Rectified Linear Unit) is widely used because it introduces non-linearity while being computationally efficient and helping reduce the vanishing gradient problem by allowing gradients to pass through unchanged for positive inputs. However, it can lead to "dead neurons" if too many outputs become zero. Sigmoid maps input values to a range between 0 and 1, making it suitable for binary classification, but it can cause vanishing gradients for very large or small inputs due to its flat tails. Tanh is similar to sigmoid but outputs values between -1 and 1, often leading to better convergence because it centers data around zero (see Table 1). However, like sigmoid, it also suffers from vanishing gradients. The choice of activation function can impact the speed of learning, stability, and ultimately the performance of the model, so it's often chosen based on the specific task and network architecture.

Table 1. Most common activation functions used in ANNs.

Function	Expression	Application
Linear	$f(z) = z$	Output layers
Logistic sigmoid	$f(z) = \dfrac{1}{1 + e^{-z}}$	Output and hidden layers
Hyperbolic tangent sigmoid	$f(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Hidden layers
ReLU	$f(z) = \max(0, z)$	Output layers

When should I use techniques like dropout, batch normalization, or regularization?: Techniques like **dropout, batch normalization,** and **regularization** are essential tools used during neural network training to improve model performance and prevent overfitting. You should use **dropout** when your model is complex or has a tendency to memorize the training data, as it randomly turns off a percentage of neurons during training, encouraging the network to learn more general and robust features. **Batch normalization** is especially helpful in deep networks because it normalizes the inputs of each layer, which can speed up training, reduce sensitivity to weight initialization, and help with vanishing or exploding gradients. It's commonly placed between the linear transformation and the activation function. **Regularization techniques**, such as **L1** and **L2 regularization**, apply penalties to large weight values, discouraging the model from becoming too complex and thus improving generalization. These methods are often used together and are most effective when you notice signs of overfitting or unstable training behavior.

Data and Evaluation

Is my data normalized/scaled correctly?: To determine if your data is normalized or scaled correctly, you should examine the range and distribution of your input features. Properly scaled data ensures that all features contribute equally during training, especially in algorithms sensitive to input magnitude like neural networks or gradient-based optimizers. If you've used **standardization**, your features should have a mean near zero and a standard deviation close to one, while normalization should transform the values to a range such as 0 to 1. You can verify this by reviewing summary statistics (like mean, min, and max) or plotting the feature distributions. It's also important to apply the same scaling transformation to both training and testing datasets to maintain consistency. Ensuring correct scaling can significantly improve model convergence and performance.

Do I have a balanced dataset? If not, how can I handle class imbalance?: To determine if you have a balanced dataset, you need to check the distribution of classes in your target variable. A balanced dataset has roughly the same number of examples for each class, while an imbalanced dataset shows a significant difference in the number of samples across classes. Class imbalance can lead to biased models that perform well on the majority class but poorly on the minority one.

If your dataset is imbalanced, you can handle it using several techniques: resampling methods like oversampling the minority class (e.g., with SMOTE) or undersampling the majority class; class weighting, which adjusts the loss function to penalize misclassifications of minority classes more heavily; or using specialized algorithms that are designed to be robust to imbalance. Evaluating model performance with metrics like precision, recall, F1-score, or AUC-ROC instead of just accuracy is also crucial in these cases.

What evaluation metrics should I use (accuracy, precision, recall, F1, etc.)?: The evaluation metrics you should use depend on the nature of your problem and the balance of your dataset. Accuracy is a common metric that measures the overall percentage of correct predictions, but it can be misleading when dealing with imbalanced datasets. In such cases, precision (the proportion of true positive predictions out of all positive predictions) and recall (the proportion of true positives out of all actual positives) provide more insight into how well your model is identifying specific classes.

F1-score, which is the harmonic mean of precision and recall, is especially useful when you need to balance both false positives and false negatives. For binary and multi-class classification problems, AUC-ROC can also be helpful to evaluate how well the model distinguishes between classes. Choosing the right metric depends on your goals—for example, in medical diagnostics, high recall may be more important to minimize missed positive cases, while in spam detection, high precision may be key to avoid false alarms.

How do training, validation, and test sets differ and why are they important?: Training, validation, and test sets serve distinct purposes in the machine learning process and are essential for building reliable models. The **training set** is used to teach the model by allowing it to learn patterns and adjust its parameters. The **validation set** is used during training to fine-tune hyperparameters and monitor the model's performance on unseen data, helping to prevent overfitting.

It acts as a checkpoint to ensure that improvements on the training data translate to better generalization. Finally, the **test set** is used only after training and validation are complete, providing an unbiased evaluation of the model's real-world performance. Keeping these sets separate ensures that the model doesn't "cheat" by memorizing the data and gives a clear picture of how well it will perform on truly new data.

Experimentation and Iteration

What assumptions am I making about my data/model?: When building a machine learning model, you often make several assumptions about your data and model, sometimes without realizing it. One common assumption is that the data is representative of the real-world scenario you're trying to model, meaning the training, validation, and test sets all come from the same distribution.

You also assume that the features used are relevant and informative, and that there's enough data to capture the underlying patterns. Depending on the algorithm, you may assume linearity, independence, or normality in the data. For example, linear regression assumes a linear relationship between features and the target, while some models assume independent and identically distributed (i.i.d.) data points. Additionally, you assume that your model architecture and hyperparameters are capable of learning the patterns without overfitting or underfitting. Recognizing these assumptions is crucial, as violating them can lead to poor performance and misleading results.

What are the key hyperparameters I can tune?: Key hyperparameters you can tune in a machine learning model include those that control the learning process, model complexity, and regularization. One of the most important is the learning rate, which determines how quickly the model updates its weights during training—a value too high can lead to instability, while too low can slow convergence. Batch size affects how many samples are used per gradient update and can influence training speed and generalization. In neural networks, you can also tune the number of layers and neurons per layer, which influence the model's capacity to learn complex patterns. Other critical hyperparameters include the activation function, dropout rate (to prevent overfitting), and regularization terms like L1 or L2 penalties. Additionally, choosing the right optimizer (such as SGD, Adam, or RMSprop) and adjusting its settings can have a major impact on performance. Hyperparameter tuning is essential for finding a good balance between underfitting and overfitting and achieving optimal model performance.

How can I systematically test different configurations?: To systematically test different configurations of your model, you can use hyperparameter tuning techniques such as grid search, random search, or more advanced methods like Bayesian optimization. Grid search involves specifying a set of possible values for each hyperparameter and training the model with every possible combination, which can be exhaustive but time-consuming. Random search, on the other hand, samples random combinations from the hyperparameter space and is often more efficient, especially when only a few parameters significantly impact performance. For even smarter tuning, Bayesian optimization builds a model of performance over the hyperparameter space and chooses new configurations based on previous results to find the best settings faster. These searches are usually done using a validation set or with cross-validation to ensure the results generalize well. Tools like scikit-learn, Optuna, or Ray Tune can automate this process, making it easier to find the optimal configuration for your model.

What insights am I getting from loss curves and metrics over epochs?: Loss curves and performance metrics plotted over epochs provide valuable insights into how well your model is learning and generalizing. By observing the training and validation loss curves, you can detect signs of underfitting, where both losses remain high, or overfitting, where the training loss decreases while the validation loss starts to rise. Ideally, both losses should decrease and then plateau, with the validation loss staying close to the training loss. Similarly, tracking metrics like accuracy, precision, recall, or F1-score over time can help you understand whether the model is improving in the areas that matter most for your task. Sudden spikes or drops may indicate issues such as noisy data, learning rate problems, or unstable training. These plots can guide decisions on when to stop training (using early stopping), adjust hyperparameters, or modify the model architecture. In essence, loss and metric trends act as real-time feedback, helping you steer model development in the right direction.

Advanced topics and Extensions

When should I use CNNs, RNNs, or Transformers instead of a standard MLP?: You should consider using CNNs, RNNs, or Transformers instead of a standard MLP when your data has specific structures or patterns that these specialized architectures are designed to capture. Convolutional Neural Networks (CNNs) are ideal for image and spatial data because they excel at detecting local features and maintaining spatial relationships through convolutional filters. Recurrent Neural Networks (RNNs) are better suited for sequential or time-series data, such as text or speech, as they process inputs in order and maintain a form of memory across time steps. However, RNNs can struggle with long sequences, which is where Transformers come in—they use attention mechanisms to capture relationships between elements regardless of their position in the sequence, making them highly effective for natural language processing, translation, and even some vision tasks. In contrast, MLPs (Multilayer Perceptrons) treat all input features as independent and are more appropriate for simpler tabular datasets without spatial or sequential structure. Choosing the right architecture depends on the nature of your input data and the patterns you want the model to learn.

How do transfer learning and fine-tuning work?: **Transfer learning** and **fine-tuning** are techniques that allow you to leverage a pre-trained model—usually trained on a large dataset like ImageNet or a massive corpus of text—to solve a different but related task with less data and computation. In transfer learning, you start by using the pre-trained model's learned features, typically by freezing its early layers (which capture general patterns like edges or grammar), and only training the final layers that are specific to your new task. This is especially useful when your dataset is small or similar in nature to the one the model was originally trained on. Fine-tuning takes this a step further: instead of just training the final layers, you unfreeze some or all of the pre-trained layers and continue training the entire model on your new data, but usually with a lower learning rate. This allows the model to adapt more precisely to the new task while still benefiting from the generalized knowledge it already possesses. Both techniques are widely used in computer vision and natural language processing, dramatically improving performance and reducing training time on new tasks.

What are some recent advancements in neural network training?: Recent advancements in neural network training have introduced innovative techniques aimed at improving efficiency, scalability, and performance. One notable development is the emergence of Relational Deep Learning, which utilizes Graph Neural Networks (GNNs) to extract meaningful patterns directly from relational databases without the need for extensive feature engineering. This approach has the potential to unlock new AI applications across various industries. In the realm of hardware optimization, researchers have proposed a novel training technique that significantly reduces the energy consumption of neural networks. This method opens the door to more sustainable AI applications by addressing the substantial energy demands typically associated with training large models. Additionally, the concept of Adaptive Class Emergence Training has been introduced to enhance neural network stability and generalization. This methodology involves progressively evolving target outputs during training, allowing networks to adapt more smoothly to complex classification tasks and reducing the risk of overfitting. Furthermore, the integration of Mixture of Experts (MoE) models has gained traction, particularly in large-scale transformer architectures. MoE models dynamically allocate computational resources to specialized subnetworks (experts), enabling efficient scaling and improved performance on diverse tasks. These advancements reflect a concerted effort to address the challenges of training neural networks, focusing on enhancing computational efficiency, scalability, and resource optimization. Ongoing research and workshops, such as the Workshop on Advancing Neural Network Training (WANT), continue to provide platforms for exploring these innovations.

What are the ethical and societal implications of deploying neural network models?: The deployment of neural network models carries significant ethical and societal implications that must be carefully considered. One major concern is bias and fairness—if the training data reflects societal biases, the model can unintentionally perpetuate discrimination in areas like hiring, lending, law enforcement, and healthcare. This can lead to unjust outcomes, particularly for marginalized groups. Another critical issue is privacy, especially when models are trained on sensitive or personal data; without proper safeguards, neural networks can memorize and potentially leak private information. Transparency and explainability also pose challenges. Neural networks, especially deep ones, often operate as "black boxes," making it difficult to understand how they reach decisions. This lack of interpretability can erode trust and make it hard to identify or correct harmful behavior. Moreover, automation and job displacement are societal concerns, as AI systems may replace human workers in certain industries, leading to economic disruptions. There's also the risk of misuse—neural networks can be we aponized for deep fakes, misinformation campaigns, surveillance, or other harmful applications. Therefore, it's crucial to approach the development and deployment of these models with strong ethical frameworks, transparent governance, and inclusive policies that prioritize fairness, accountability, and human oversight.

CONCLUSION

As artificial neural networks (ANNs) continue to drive advancements in machine learning and artificial intelligence, it becomes increasingly important for learners to engage critically and thoughtfully with the concepts, tools, and challenges involved. This review has outlined a range of key questions that aspiring ANN practitioners should ask themselves throughout their learning journey—from foundational understanding and architectural choices to training strategies, evaluation metrics, and ethical implications. By consistently reflecting on these questions, learners can deepen their comprehension, avoid common pitfalls, and develop more robust, responsible, and effective models. Importantly, such a reflective approach fosters not only technical competence but also an awareness of the broader impact of neural network applications in real-world contexts. Ultimately, asking the right questions is not just a learning strategy—it is a mindset that distinguishes skilled, thoughtful practitioners in the evolving field of neural networks.

Key Points

- This paper emphasizes the importance of questioning how closely Artificial Neural Networks should model the human brain.
- A core focus is placed on the mechanisms of forward and backward propagation—key processes in how ANNs learn.
- Finally, it urges ANN developers to ask targeted questions about the training process, optimization strategies (e.g., gradient descent variants), and the impact of iteration.

ACKNOWLEDGEMENT

I would like to thank Prof. Jane Alam sir for encouraging me to write this article.

REFERENCES

Alkinani HH, Al-Hameedi ATT, Dunn-Norman S. 2020. Artificial neural network models to predict lost circulation in natural and induced fractures. SN Applied Sciences. 2:1-13. https://doi.org/10.1007/s42452-020-03827-3.

Amari SI. 1993. Backpropagation and stochastic gradient descent method. Neurocomputing, 5(4-5): 185-196. doi.org/10.1016/0925-2312(93)90006-O.

Anuraganand S. 2021. Guided parallelized stochastic gradient descent for delay compensation. Applied Soft Computing. 102:107084. doi.org/10.1016/j.asoc.2021.107084.

Benjamin AS, Fernandes HL, Tomlinson T, Ramkumar P, VerSteeg C, Chowdhury RH, Miller LE, Kording KP. 2018. Modern Machine Learning as a Benchmark for Fitting Neural Responses. Front. Comput. Neurosci. 12, Article 56, p. 1-13. doi: 10.3389/fncom.2018.00056.

Broad A, Arkin J, Ratliff N, Howard T, Argall B. 2017. Real-time natural language corrections for assistive robotic manipulators. The International Journal of Robotics Research. 36(5-7): 684-698. doi.org/10.1177/0278364917706418.

Bruno G, Antonelli D, Stadnicka D. 2021. Evaluating the effect of learning rate, batch size and assignment strategies on the production performance. Journal of Industrial and Production Engineering. 38(2): 137-147. doi.org/10.1080/21681015.2021.1883133.

Chib P S, Singh P. 2024. Recent advancements in end-to-end autonomous driving using deep learning: A survey. IEEE Transactions on Intelligent Vehicles, 9(1): 103-118. doi: 10.1109/TIV.2023.3318070.

Crone SF. 2002. Training artificial neural networks for time series prediction using asymmetric cost functions. In

Proceedings of the 9th International Conference on Neural Information Processing. ICONIP'02. IEEE. Singapore, vol. 5. p. 2374-2380. doi: 10.1109/ICONIP.2002.1201919.

D'Amour A. *et al*. 2022. Underspecification Presents Challenges for Credibility in Modern Machine Learning. The Journal of Machine Learning Research. 23(226):1−61.

Defernez M, Kemsley EK. 1999. Avoiding overfitting in the analysis of high-dimensional data with artificial neural networks (ANNs). Analyst. 124(11): 1675-1681. doi: 10.1039/A905556H.

Dubey SR, Singh SK, Chaudhuri BB. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing. 503: 92-108. doi.org/10. 1016/j .neucom.2022.06.111.

Gardner WA. 1984. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. Signal processing, 6(2): 113-133. doi.org/10.1016/0165-1684(84)90013-6.

Gerven MV, Bohte S. 2017. Artificial neural networks as models of neural information processing. Frontiers in Computational Neuroscience. 11: 114. doi.org/ 10.3 389/ fncom.2017.00114.

Granziol D, Zohren S, Roberts S. 2022. Learning rates as a function of batch size: A random matrix theory approach to neural network training. Journal of Machine Learning Research, 23(173): 1-65.

Hawkins DM. 2004. The problem of overfitting. Journal of chemical information and computer sciences. 44(1): 1-12. doi: 10.1021/ci0342472.

He F, Liu T, Tao D. 2019. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. Advances in neural information processing systems. Curran Associates, Inc. vol. 32.

Hu C, Shi Q, Liu L, Wejinya U, Hasegawa Y, Shen Y. 2017. Robotics in biomedical and healthcare engineering. Journal of healthcare engineering. 2017: 1610372-1610373. doi: 10.1155/2017/1610372.

Jafarian A, Nia SM, Golmankhaneh AK, Baleanu D. 2018. On artificial neural networks approach with new cost functions. Applied Mathematics and Computation, 339: 546-555. doi.org/10.1016/j.amc.2018.07.053.

Khan M, Hussain T, Ullah H, Ser JD, Rezaei M, Kumar N, Hijji M, Bellavista P, de Albuquerque VHC. 2022. Vision-based semantic segmentation in scene understanding for autonomous driving: Recent achievements, challenges, and outlooks. IEEE Transactions on Intelligent Transportation Systems. 23(12): 22694-22715. doi: 10.1109/TITS.2022.3207665.

Kingma DP, Ba J. 2014. Adam: A method for stochastic optimization. arXiv prep. arXiv:1412.6980. doi.org/10.48550/arXiv.1412.6980.

Loce RP, Bernal EA, Wu W, Bala R. 2013. Computer vision in roadway transportation systems: a survey. Journal of Electronic Imaging, 22(4): 041121-041121. doi.org/10.1117/1.JEI.22.4.041121.

Matel E, Vahdatikhaki F, Hosseinyalamdary S, Evers T, Voordijk H. 2022. An artificial neural network approach for cost estimation of engineering services. International journal of construction management. 22(7): 1274-1287. doi.org/10.1080/15623599.2019.1692400.

Piotrowski AP, Napiorkowski JJ. 2013. A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling. Journal of Hydrology. 476: 97-111. doi.org/10.1016/ j.jhydrol. 2012.10.019.

Pothuganti S. 2018. Review on over-fitting and under-fitting problems in Machine Learning and solutions. Int. J. Adv. Res. Electr. Electron. Instrum. Eng. 7(9): 3692-3695.

Prasad PS, Gunjan VK, Pathak R, Mukherjee S. 2021. Applications of artificial intelligence in biomedical engineering. In Handbook of Artificial Intelligence in Biomedical Engineering. 1st Edition. Apple Academic Press. p. 125-145. doi.org/10.1201/9781003045564.

Prieto A, Prieto B, Ortigosa EM, Ros E, Pelayo F, Ortega J, Rojas I. 2016. Neural networks: An overview of early research, current frameworks and new challenges. Neurocomputing. 214: 242-268. doi.org/10.1016/j. neucom.2016.06.014.

Rasamoelina AD, Adjailia F, Sinčák P. 2020. A review of activation function for artificial neural network. In 2020 IEEE 18th world symposium on applied machine intelligence and informatics (SAMI). IEEE. Herlany, Slovakia. p.281-286. doi: 10.1109/SAMI 48414.20 20.9108717.

Santos CFGD, Papa JP. 2022. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. ACM Computing Surveys (CSUR). 54(10s):1-25. doi.org/10.1145/351041.

Sarkar C, Das B, Rawat VS, Wahlang JB, Nongpiur A, Tiewsoh I, Lyngdoh NM, Das D, Bidarolli M, Sony HT. 2023. Artificial Intelligence and Machine Learning Technology Driven Modern Drug Discovery and Development. Int. J. Mol. Sci. 24: 2026. https://doi.org/10.3390/ijms24032026.

Schaffer C. 1993. Overfitting avoidance as bias. Machine learning. 10: 153-178. doi.org/10.1007/BF00993504.

Sharma S, Sharma S, Athaiya A. 2017. Activation functions in neural networks. Towards Data Sci. 6(12): 310-316.

Sivaraman S, Trivedi MM. 2013. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. IEEE transactions on intelligent transportation systems. 14(4): 1773-1795. doi: 10.1109/TITS.2013.2266661.

Smith SL, Kindermans PJ, Ying C, Le QV. 2017. Don't decay the learning rate, increase the batch size. arXiv prep. arXiv:1711.00489. doi.org/10.48550/arXiv.1711.00489.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research. 15(1): 1929-1958.

Varshney M, Singh P. 2021. Optimizing nonlinear activation function for convolutional neural networks. Signal, Image and Video Processing. 15(6): 1323-1330. https://doi.org/10.1007/s11760-021-01863-z.

Ying X. 2019. An overview of overfitting and its solutions. J. Phys.: Conf. Ser. IOP Publishing. 1168: 022022. doi 10.1088/1742-6596/1168/2/022022.

Zhou B, Yang G, Shi Z, Ma S. 2022. Natural language processing for smart healthcare. IEEE Reviews in Biomedical Engineering. 17: 4-18. doi: 10.1109/RBME.2022.3210270.

Zhou X, Zhang W, Chen Z, Diao S, Zhang T. 2021. Efficient neural network training via forward and backward propagation sparsification. Advances in neural information processing systems, 34: 15216-15229.

Zou F, Shen L, Jie Z, Zhang W, Liu W. 2019. A sufficient condition for convergences of adam and rmsprop. In Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition. California. p. 11127-11135.

****