International Journal of Operations Research and Artificial Intelligence

www.ijorai.reapress.com

Int. J. Oper. Res. Artif. Intell . Vol. 1, No. 3 (2025) 139–147.

Paper Type: Original Article

Determining the Optimal Neural Network Architecture and Parameters: A Comprehensive Review and Practical Manual for Data Scientists

Tajedin Derikvand

Department of Mathematics, Marvdasht Branch, Islamic Azad University, Marvdasht, Iran; teloori@yahoo.com.

Citation:

Received: 5 April 2024 *Revised: 27 June 2024* *Accepted: 14 Auguste 2024*	Derikvand, T. (2025). Determining the optimal neural network architecture and parameters: A comprehensive review and practical manual for data scientists. *International journal of operations research and artificial intelligence, 1*(3), 139-147.

Abstract

The performance of a Neural Network (NN) critically depends on its architecture and parameter configuration. Selecting optimal network depth, width, activation functions, and training hyperparameters is a nontrivial task that significantly influences convergence speed, generalization, and computational efficiency. Despite significant progress in optimization and Auto Machine Learning (ML) research, a unified practical framework for determining the optimal NN structure for a given dataset remains elusive. This paper presents a comprehensive review of methods for NN optimization, organized across two hierarchical levels: inner-level parameter optimization (weights, biases, learning rates) and outer-level architecture optimization (depth, width, activation functions, and other hyperparameters). We systematically examine manual design heuristics, search-based techniques (grid, random, Bayesian), evolutionary and reinforcement-based Neural Architecture Search (NAS), differentiable NAS, and meta-learning approaches. Emerging trends such as differentiable hyperparameter optimization, continuous architecture representations, and operator-based frameworks are also discussed. In addition to the theoretical synthesis, the paper provides a practical manual for data scientists, detailing a step-by-step workflow for designing, training, and refining NNs efficiently. The review concludes with an analysis of current challenges and outlines future research directions toward mathematically grounded, unified frameworks for network optimization.

Keywords: Neural network, Architecture optimization, Hyperparameter tuning, Neural architecture search, Auto machine learning, Meta-learning.

1 | Introduction

Neural Networks (NNs) have become the dominant modeling framework across a wide range of scientific and engineering domains, from computer vision and natural language processing to physics-informed learning and control [1–3]. The success of these models depends heavily on the architecture and hyperparameters chosen before training [4–6]. However, determining the optimal configuration of these components remains one of the most challenging and least theoretically resolved problems in deep learning [1].

Corresponding Author: teloori@yahoo.com

https://doi.org/10.48314/ijorai.v1i3.73

An NN's architecture defines its functional capacity—the number of layers (depth), the number of neurons per layer (width), the type of activation functions, and the pattern of interconnections. Its training parameters—including learning rate, batch size, and regularization coefficients—govern how effectively this capacity can be exploited to minimize the chosen loss function. Because these elements interact in complex and nonlinear ways, even small design changes can lead to dramatic differences in training behavior and generalization performance [1], [2].

The problem of determining an optimal NN can be viewed as a bi-level optimization task: at the inner level, the model learns optimal parameters (weights and biases) for a fixed architecture; at the outer level, the architecture itself is optimized to minimize validation error [4–6]. The outer-level optimization is discrete, highly non-convex, and computationally expensive—often requiring training and evaluating thousands of candidate networks. Consequently, a diverse set of strategies has emerged, ranging from manual trial-and-error and heuristic rules to fully automated optimization frameworks such as Bayesian optimization, evolutionary algorithms, and Neural Architecture Search (NAS) [7], [8].

Despite the vast literature, the field lacks a unified synthesis that connects theoretical underpinnings with practical implementation guidelines. Data scientists and applied researchers often rely on ad hoc experimentation without a clear methodological direction. This review aims to bridge this gap by offering both a conceptual taxonomy of existing approaches and a practical manual outlining how these techniques can be applied systematically to real-world datasets.

The main contributions of this review are threefold:

I. A comprehensive categorization of NN optimization strategies, encompassing both parameter and architecture determination.

II. A unified mathematical formulation of the optimal NN design problem as a bi-level optimization process.

III. A practitioner-oriented workflow, providing step-by-step guidance for data scientists to design, tune, and evaluate NNs effectively.

The remainder of this paper is organized as follows. Section 2 presents the theoretical foundations and formalizes the bi-level optimization framework. Within this section, Subsection [sec: challenges] discusses the main challenges in determining optimal NNs. Section 3 reviews parameter optimization techniques, including gradient-based and adaptive methods. Subsection [sec:weight_init] details weight initialization strategies, while Subsection [sec: practical] provides practical recommendations and presents recommended network configurations based on dataset type and size.

2 | Background and Theoretical Foundations

2.1 | Fundamentals of Neural Networks

A NN is a parametric function $f_\theta: \mathbb{R}^n \to \mathbb{R}^m$ defined by a composition of affine transformations and nonlinear activations:

$$f_\theta(x) = \phi_L(W_L\phi_{L-1}(W_{L-1}\cdots\phi_1(W_1x + b_1) + b_{L-1}) + b_L),$$

where $\theta = \{W_1, \dots, W_L, b_1, \dots, b_L\}$ denotes all trainable parameters, L is the network depth, and each ϕ_i is a nonlinear activation function. The training process aims to minimize an empirical loss function [1]:

$$\mathcal{L}_{train}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\,(f_\theta(x_i), y_i),$$

where (x_i, y_i) are samples from the dataset.

The capacity of a NN is primarily determined by:

I. Depth: number of layers (hierarchical representation power),

II. Width: number of neurons per layer (feature diversity),

III. Activation functions: shape of the nonlinear mapping [9],

IV. Regularization: constraints to avoid overfitting [6], [10].

The optimization landscape of $\mathcal{L}_{\text{train}}$ is typically non-convex and high-dimensional, posing theoretical and computational challenges [11], [12].

2.2 | Levels of Optimization

NN optimization can be formalized as a bi-level optimization problem consisting of:

$$\begin{cases} \theta^*(\alpha) = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{\text{train}} \left(f_\theta^{(\alpha)}(X_{\text{train}}), Y_{\text{train}} \right), \\ \alpha^* = \underset{\alpha}{\operatorname{argmin}} \mathcal{L}_{\text{val}} \left(f_{\theta^*(\alpha)}^{(\alpha)}(X_{\text{val}}), Y_{\text{val}} \right), \end{cases}$$

where:

I. θ: model parameters (weights, biases),

II. α: architectural and hyperparameter configuration,

III. $\mathcal{L}_{\text{train}}$: training loss,

IV. $\mathcal{L}_{\text{val}}$: validation loss.

The inner level corresponds to parameter learning, typically solved using gradient-based methods such as Stochastic Gradient Descent (SGD) [8] or Kingma [11], while the outer level searches over architectures and hyperparameters to find configurations that generalize best to unseen data [4], [6].

2.3 | Challenges in Determining Optimal Neural Networks

Determining the optimal NN configuration involves several theoretical and practical challenges. These challenges stem from the intricate interplay between architecture, optimization dynamics, data distribution, and computational limits [1], [2]. Below, we summarize the main issues and propose practical or theoretical approaches to mitigate them.

I. Non-convexity and multimodality: the loss landscape of deep networks contains many local minima, saddle points, and flat regions. This non-convexity complicates convergence analysis and makes theoretical guarantees rare.

II. Solutions: techniques such as SGD with momentum, adaptive learning rates, and noise injection (e.g., dropout or Batch Normalization (BN)) help escape saddle points [13]. Modern analysis shows that many local minima are "good" (i.e., have similar generalization error), implying that optimization need not find the global minimum to achieve optimal performance. Additionally, landscape smoothing methods and second-order approximations improve stability.

III. Discrete and continuous variables: NN optimization involves both continuous variables (weights, learning rates) and discrete architectural decisions (number of layers, neurons, and activation types). This hybrid nature complicates direct optimization.

IV. Solutions: NAS and differentiable NAS convert discrete search spaces into continuous relaxations, enabling gradient-based optimization. Alternatively, evolutionary algorithms and reinforcement learning have been successfully used to efficiently handle discrete hyperparameter search.

V. Computational cost: evaluating each candidate architecture typically requires full or partial training, which is computationally prohibitive.

VI. Solutions: multi-fidelity optimization, surrogate modeling, and early-stopping heuristics (e.g., warm restarts or cyclical learning rates) significantly reduce computational demand. Weight-sharing strategies in NAS and meta-learning approaches reuse knowledge from previous searches to accelerate convergence.

VII. Data dependency: the optimal architecture often depends on dataset characteristics such as sample size, noise level, and intrinsic dimensionality. No single model universally performs best (the "no free lunch" principle) [14].

VIII. Solutions: meta-learning and transfer learning frameworks adapt architectures to new datasets by leveraging priors from related tasks. Data-driven model scaling rules, such as EfficientNet's compound scaling, provide a systematic way to adapt to data size and complexity.

IX. Generalization vs. overfitting: deep networks with large capacity easily overfit small or noisy datasets. Conversely, too much regularization may lead to underfitting.

X. Solutions: regularization techniques such as dropout, BN, and learning rate averaging help balance the trade-off.

XI. Reproducibility and transferability: hyperparameter sensitivity and stochastic training procedures often lead to results that vary across runs, hindering reproducibility. Moreover, architectures optimized on one dataset may not generalize to another.

XII. Solutions: systematic search frameworks, cyclical and restart-based schedules, and averaging techniques improve robustness and transferability [15].

XIII. Interpretability and theoretical understanding: the relationship between architectural design and learning dynamics remains only partially understood. This lack of interpretability limits principled optimization.

XIV. Solutions: information-theoretic analyses, dynamical systems modeling, and operator-theoretic frameworks provide emerging theoretical tools [1]. These approaches aim to connect optimization trajectories and generalization behavior with underlying mathematical structures.

In summary, although the optimal design of NNs remains a complex problem, modern approaches—combining stochastic optimization, differentiable relaxation, and meta-learning—are steadily closing the gap between empirical practice and theoretical understanding. These developments form the foundation for automated, scalable, and interpretable neural architecture optimization, as explored in later sections.

3 | Optimization of Parameters

The optimization of parameters—namely the weights and biases of a NN—constitutes the inner level of the bi-level design problem. It directly determines how effectively a given architecture learns from data and generalizes to unseen samples. While the architecture fixes the function space that can be represented, parameter optimization determines the particular function within that space that minimizes the empirical loss. This section reviews significant developments and practical methods in parameter optimization.

3.1 | Formulation of the Parameter Optimization Problem

Given a fixed architecture with parameters θ, the learning task is to minimize the empirical loss:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{\text{train}}(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(f_\theta(x_i), y_i\right),$$

where $\ell(\cdot,\cdot)$ is a differentiable loss function. The optimization landscape of $\mathcal{L}_{\text{train}}(\theta)$ is non-convex, high-dimensional, and often ill-conditioned, leading to complex convergence behaviors.

The general update rule for iterative optimization is given by:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}_{\text{train}}(\theta_t),$$

where η_t is the learning rate at iteration t. The choice of optimization algorithm, learning rate schedule, initialization, and regularization strategy all significantly affect convergence and generalization.

3.2 | Gradient-Based Optimization Algorithms

Most NN training methods are variants of SGD or its adaptive extensions. These algorithms approximate the true gradient using small batches of samples, improving computational efficiency.

3.2.1 | Stochastic gradient descent

The SGD algorithm updates parameters using mini-batch gradients:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}_{\text{train}}(\theta_t),$$

where $B_t \subset \{1, \dots, N\}$ is a randomly sampled mini-batch. Despite its simplicity, SGD often achieves excellent generalization, especially with proper momentum and learning rate scheduling.

3.2.2 | Momentum and Nesterov acceleration

Momentum accelerates SGD by accumulating past gradients:

$$v_{t+1} = \mu v_t - \eta_t \nabla_\theta \mathcal{L}_{B_t}(\theta_t), \quad \theta_{t+1} = \theta_t + v_{t+1},$$

where $\mu \in [0,1)$ controls the contribution of previous updates. Nesterov Accelerated Gradient (NAG) further anticipates future updates by evaluating the gradient at a look-ahead point:

$$v_{t+1} = \mu v_t - \eta_t \nabla_\theta \mathcal{L}_{B_t}(\theta_t + \mu v_t).$$

Momentum-based methods remain central in large-scale deep learning, particularly when combined with cyclical or restart-based learning rate schedules.

3.2.3 | Adaptive methods: AdaGrad, root mean square prop, and Adam

Adaptive algorithms scale the learning rate individually for each parameter based on historical gradient magnitudes. AdaGrad accumulates the squared gradients:

$$g_t = g_{t-1} + \left(\nabla_\theta \mathcal{L}_{B_t}(\theta_t)\right)^2, \quad \theta_{t+1} = \theta_t - \eta \frac{\nabla_\theta \mathcal{L}_{B_t}(\theta_t)}{\sqrt{g_t + \epsilon}}.$$

While effective for sparse features, AdaGrad's monotonically decreasing learning rate can lead to premature convergence. Root mean square Prop mitigates this by using an exponential moving average of squared gradients:

$$g_t = \rho g_{t-1} + (1-\rho)\left(\nabla_\theta \mathcal{L}_{B_t}(\theta_t)\right)^2.$$

Adam combines momentum and RMSProp, maintaining first and second moment estimates:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}_{B_t}(\theta_t), \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}_{B_t}(\theta_t)\right)^2, \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\ \theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \end{aligned}$$

Adam has become the de facto optimizer in many modern architectures due to its stability and low tuning requirements. However, recent studies suggest that SGD with momentum generalizes better on large-scale tasks.

3.2.4 | Second-order and Quasi-Newton methods

While gradient-based methods use first-order information, second-order methods exploit curvature (the Hessian) for faster convergence. The Newton update rule is:

$$\theta_{t+1} = \theta_t - H_t^{-1}\nabla_\theta \mathcal{L}(\theta_t),$$

where H_t is the Hessian matrix. Exact computation is infeasible for large models, leading to approximations such as BFGS, L-BFGS, and K-FAC. These methods are particularly effective for fine-tuning or small networks.

4 | Weight Initialization Strategies

The initialization of weights significantly affects the training dynamics. Poor initialization can cause vanishing or exploding gradients, leading to divergence or slow convergence. Let $W^{(l)}$ denotes the weight matrix of layer l. Common initialization schemes include [16]:

Xavier/Glorot initialization

$$W^{(l)} \sim \mathcal{U}\left[-\sqrt{\frac{6}{n_l + n_{l+1}}}, \sqrt{\frac{6}{n_l + n_{l+1}}}\right],$$

Designed for tanh and sigmoid activations to maintain constant variance through layers [9]. It initialization:

$$W^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_l}\right),$$

Optimized for ReLU activations to prevent vanishing gradients.

Orthogonal initialization

Ensures weight matrices are orthogonal to preserve information flow across layers.

4.1 | Learning Rate Scheduling

The learning rate η_t critically determines training stability and convergence. Static learning rates often fail to adapt to varying gradient magnitudes, leading to oscillations or stagnation. Common scheduling strategies include:

I. Step decay: reduce η_t by a fixed factor every k epochs.

II. Exponential decay: $\eta_t = \eta_0 e^{-\lambda t}$.

III. Cosine annealing: smoothly decreases η_t using a cosine function.

IV. Cyclical learning rate: periodically increases and decreases the learning rate to escape local minima.

V. One-cycle policy: increases η_t initially, then gradually reduces it, improving both speed and generalization.

4.2 | Regularization Techniques

Regularization prevents overfitting by penalizing model complexity or introducing stochasticity during training.

4.2.1 | Weight decay and norm penalties

The most common regularization method adds an L_2 penalty:

$$\mathcal{L}' = \mathcal{L}_{\text{train}} + \lambda \parallel \theta \parallel_2^2,$$

where $\lambda > 0$ controls the strength of regularization. L_1 penalties encourage sparsity [11].

4.2.2 | Dropout

Dropout randomly deactivates neurons during training:

$$h_i^{(l)} = r_i^{(l)} \phi\left(W^{(l)} h^{(l-1)} + b^{(l)}\right),$$

where $r_i^{(l)} \sim \text{Bernoulli}(p)$. This prevents co-adaptation of neurons and improves generalization.

4.2.3 | Batch normalization

BN stabilizes learning by normalizing layer inputs:

$$\hat{h}^{(l)} = \frac{h^{(l)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y^{(l)} = \gamma \hat{h}^{(l)} + \beta,$$

where μ_B and σ_B^2 are the mini-batch mean and variance. BN improves gradient flow and allows higher learning rates [10].

4.3 | Early Stopping and Model Averaging

Early stopping halts training when the validation loss stops improving, mitigating overfitting. Model averaging techniques such as Stochastic Weight Averaging (SWA) combine weights from multiple epochs to produce flatter minima and improve generalization [15].

4.4 | Practical Recommendations

From a practitioner's standpoint, the following guidelines are widely effective for parameter optimization and architecture selection:

Weight initialization

Use the initialization for ReLU-based networks and Xavier/Glorot initialization for sigmoid or tanh activations. For recurrent architectures (e.g., RNNs and LSTMs), consider orthogonal initialization to preserve long-term dependencies.

Optimizer selection

Start with Adam for fast convergence and low tuning requirements. For extensive datasets or convolutional networks, SGD with momentum may yield better generalization.

Learning rate scheduling

Use adaptive schedules such as cosine annealing or the one-cycle policy to improve both convergence speed and generalization.

Regularization

Combine techniques like weight decay, dropout, and BN. Adjust dropout rates depending on dataset size (higher for small datasets, lower for large datasets).

Early stopping and model averaging

Monitor validation loss and employ early stopping. For more stable generalization, use techniques like SWA.

Architecture adaptation to data type

Tailor depth, width, and activation functions based on the characteristics of the dataset (image, text, tabular, time series) and its size.

Hyperparameter tuning workflow

I. Start with baseline hyperparameters from the literature for similar tasks.

II. Perform coarse search over learning rate, batch size, and number of layers.

III. Refine using Bayesian optimization, Hyperband, or grid/random search.

IV. Re-evaluate with validation data and iterate.

Recommended network configurations based on data type and size

When designing network configurations, the optimal setup heavily depends on data type (e.g., real-time, bulk transfer, sensitive data) and data size (small, medium, large).

Table 1. Practical guidance for NN architecture selection based on dataset type and size, including medical datasets.

Data Type	Dataset Size	Depth (Layers)	Width (Neurons/Layer)	Activation
Image (e.g., CIFAR, ImageNet)	Small (< 10k)	3-5	64-128	ReLU / Leaky ReLU
	Medium ($10k - 1M$)	10-20	128-512	ReLU / Leaky ReLU
	Large ($> 1M$)	20-50	256-1024	ReLU / GELU
Text / NLP (sequence data)	Small	2-3 (RNN/LSTM layers)	128-256	Tanh / ReLU
	Medium	3-6	256-512	ReLU / GELU
	Large	6-12	512-1024	GELU / ReLU
Tabular/structured data	Small	2-3	32-128	ReLU / Leaky ReLU
	Medium	3-5	64-256	ReLU / Leaky ReLU
	Large	5-10	128-512	ReLU / GELU
Time series/forecasting	Small	1-2 (RNN/GRU/LSTM)	64-128	Tanh / ReLU
	Medium	2-4	128-256	ReLU / GELU
	Large	4-8	256-512	ReLU / GELU
Medical imaging (MRI, CT, X-ray)	Small ($< 5k$)	3-5	32-128	ReLU/Leaky ReLU
	Medium (5k-50k)	8-20	128-512	ReLU / GELU
	Large ($> 50k$)	20-50	256-1024	ReLU / GELU
Genomic / high-dimensional omics data	Small (< 1k)	2-3	64-128	ReLU / Leaky ReLU
	Medium ($1k - 10k$)	3-6	128-256	ReLU / GELU
	Large ($> 10k$)	6-12	256-512	ReLU / GELU
Electronic health records / tabular medical data	Small	2-3	32-128	ReLU / Leaky ReLU
	Medium	3-5	64-256	ReLU / GELU
	Large	5-10	128-512	ReLU / GELU

These are heuristic recommendations and should be validated empirically.

Conflict of Interest Disclosure

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial or non-financial interest in the subject matter discussed in this manuscript.

Data Availability Statement

The datasets used and/or analyzed during the current study are not publicly available due to [reason if applicable], but can be made available by the corresponding author when scientifically justified.

Funding Statement

The authors confirm that no financial support was provided for the research, authorship, or publication of this article.

Reference

[1] Tan, M., & Le, Q. (2019). EfficientNet: rethinking model scaling for convolutional neural networks. *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 6105–6114). PMLR. https://proceedings.mlr.press/v97/tan19a.html?ref=ji

[2] Rajpurkar, P., Chen, E., Banerjee, O., & Topol, E. J. (2022). AI in health and medicine. *Nature medicine*, *28*(1), 31–38. https://doi.org/10.1038/s41591-021-01614-0

[3] Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2017). Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE journal of biomedical and health informatics*, *22*(5), 1589–1604. https://doi.org/10.1109/JBHI.2017.2767063

[4] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: a novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, *18*(185), 1–52. http://jmlr.org/papers/v18/16-558.html

[5] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. mach. learn. res.*, *13*(null), 281–305. https://dl.acm.org/doi/pdf/10.5555/2188385.2188395

[6] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* (Vol. 25, pp. 1–9). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[7] Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 ieee winter conference on applications of computer vision (WACV)* (pp. 464–472). IEEE. https://doi.org/10.1109/WACV.2017.58

[8] Gao, Z., & Wang, X. (2019). Deep learning. In Hu, L. & Zhang, Z. (Eds.), *EEG signal processing and feature extraction* (pp. 325–333). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-13-9113-2_16

[9] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. mach. learn. res.*, *15*(1), 1929–1958. https://dl.acm.org/doi/pdf/10.5555/2627435.2670313

[10] Martens, J., & Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. *Proceedings of the 32nd international conference on machine learning* (Vol. 37, pp. 2408–2417). Lille, France: PMLR. https://proceedings.mlr.press/v37/martens15.html

[11] Kingma, D. P. (2014). *Adam: A method for stochastic optimization*. https://doi.org/10.48550/arXiv.1412.6980

[12] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR. https://proceedings.mlr.press/v9/glorot10a.html

[13] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7), 1–39. https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems* (Vol. 30, pp. 1–11). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[15] Loshchilov, I., & Hutter, F. (2016). *Sgdr: stochastic gradient descent with warm restarts*. https://doi.org/10.48550/arXiv.1608.03983

[16] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: surpassing human-level performance on imagenet classification. *Proceedings of the ieee international conference on computer vision* (pp. 1026–1034). IEEE International Conference on Computer Vision. https://openaccess.thecvf.com/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf