TIG Neural Network Gradient Descent Challenge

The Innovation Game Labs

December, 2025

1 Introduction

The recent surge in Artificial Intelligence (AI) has been largely driven by deep learning [1, 2, 3, 4], a movement resulting from the combination of large datasets, highly parallelized compute, and rich neural network architectures with automatic differentiation. Central to deep learning progress is Stochastic Gradient Descent (SGD) and related algorithms for neural network training [5, 6, 7], which apply some form of the first-order gradient in an iterative manner to the network parameters. In particular, the Adam optimiser [8] is currently one of the most cited papers of the decade, accumulating over 190 thousand citations.

To encourage the discovery of novel first-order gradient descent optimisation algorithms capable of effectively optimizing deep neural networks, we introduce the Neural Network Gradient Descent challenge at TIG. This challenge aims to emulate the deep learning setting by training multilayer perceptrons (MLPs), the simplest foundational neural network architecture [9], on a nonlinear regression task, resulting in loss functions with typical deep learning characteristics such as:

- **Very large dimensionality** with for example moderate MLP sizes easily reaching $O(10^6)$ parameters.
- **Many local minima and saddle points** due to the excessive overparameterizarion of neural networks from the classical perspective of statistics [10, 11].
- **The phenomenon of overfitting** where many training loss minima correspond to networks with poor performance on unseen data points.

2 Problem Description

Given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_i$ split into train $\mathcal{D}_{\text{train}}$, validation $\mathcal{D}_{\text{val}}$ and test $\mathcal{D}_{\text{test}}$. The task is to use the $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$ to train a standard MLP architecture [9] $\hat{f}_{\mathbf{w}} : \mathbb{R}^D \to \mathbb{R}$. Such that the Mean Squared Error on the test data is minimized

$$\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \|y_i - \hat{f}_{\mathbf{w}}(\mathbf{x}_i)\|^2.$$

3 TIG Challenge

To focus innovation on the optimiser component of the training algorithm, the Neural Network Gradient Descent challenge has a different overall structure from most current TIG challenges in two key aspects:

1. The optimiser algorithm submission does not output a “solution”, in the sense that it is embedded inside a “parent” algorithm called the training loop that uses the optimiser iteratively to compute the solution.

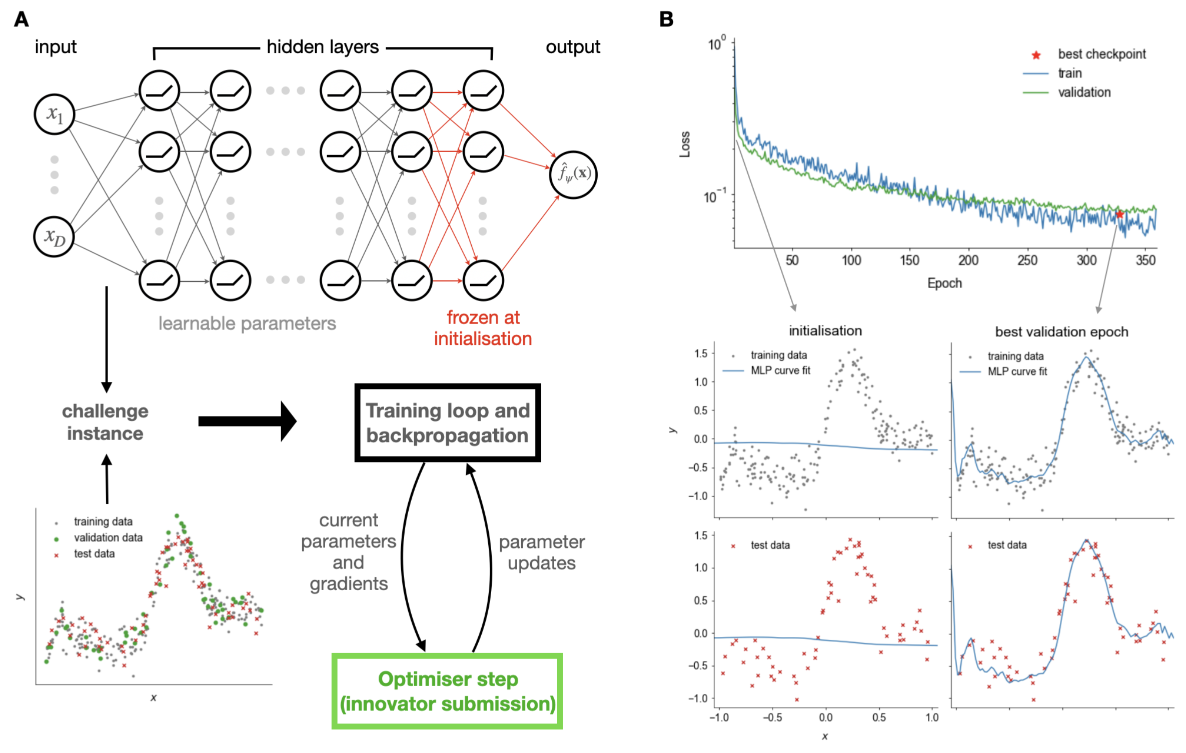

Figure 1: **Schematic of the TIG Neural Network Training challenge.** **(A)** Here we show the MLP architecture (top), an example 1D dataset generated with a GP (bottom left), and a high-level visualization of the challenge training loop structure (bottom right). **(B)** Example training of an MLP with 2 hidden layers, showing train and validation loss curves (top) and the corresponding MLP functions at initialisation (bottom left) and after early stopping (bottom right).

2. The optimiser algorithm is ran many times for a single challenge instance, with inputs and outputs constrained to the structure of the training loop and determined by intermediate states.

High-level structure A challenge instance is defined by three main components:

- A dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_i^N$ that is randomly generated by adding white noise $\xi_i \sim \mathcal{N}(0, \sigma^2_{\text{data}})$ to a random smooth function $f : \mathbb{R}^D \to \mathbb{R}$ constructed using

$$y_i = f(\mathbf{x}_i) + \xi_i, \quad f(\mathbf{x}) = \mathbf{a} \cdot \boldsymbol{\phi}(\mathbf{x}), \quad \mathbf{a} \sim \mathcal{N}(\mathbf{0}_K, \mathbf{I}_K), \tag{1}$$

  evaluated at uniform random locations in the unit hypercube $\mathbf{x}_i \in [-1, 1]^D$. Here $\boldsymbol{\phi}$ is built using Random Fourier Features and K is the number of randomised basis functions.

  The dataset $\mathcal{D}$ is furthermore split into train $\mathcal{D}_{\text{train}}$, validation $\mathcal{D}_{\text{val}}$ and test $\mathcal{D}_{\text{test}}$ sets of sizes $N_{\text{train}} = 1000$, $N_{\text{val}} = 200$ and $N_{\text{test}} = 250$.

- A standard MLP architecture $\hat{f}_{\mathbf{w}} : \mathbb{R}^D \to \mathbb{R}$ with randomly initialized parameters $\mathbf{w}$ where its hidden layers all share the same specified width and contain ReLU-BatchNorm activation functions.

- A mean squared error (MSE) loss function between some set of targets $\{y_i\}$ and MLP outputs $\{\hat{f}_{\mathbf{w}}(\mathbf{x}_i)\}$

$$\mathcal{L}(\mathbf{w}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{f}_{\mathbf{w}}(\mathbf{x}_i)\|^2, \tag{2}$$

  which is used during train, validation and test evaluations.

Details of the random instance generation are given in Section 4. The train set is then divided into B batches $\mathcal{D}_{\text{train}} \to \{\mathcal{D}_{\text{batch }b}\}_1^B$ of size N_{batch} that are then fed into the training loop, where each loop iteration (epoch) consists of:

1. Sampling batches in random order, where for every batch:
    - Compute the parameter location $\tilde{\mathbf{w}}$ (which can be different to current $\mathbf{w}$ like in Nesterov momentum [12]) at which we evaluate the training loss gradients
    - Compute the regression loss $\mathcal{L}$ and its gradients $\mathbf{g} = \nabla_{\mathbf{w}} \mathcal{L}(\tilde{\mathbf{w}}; \mathcal{D}_{\text{batch}})$ using a forward-and-backward pass [5] through the MLP
    - Run one optimiser step to transform $\mathbf{g}$ into parameter updates $\mathbf{u}$
    - Apply updates $\mathbf{w} \to \mathbf{w} + \mathbf{u}$

    Note that multiple gradient steps are applied on different subsets (batches) of $\mathcal{D}_{\text{train}}$ per epoch, hence the term "stochastic" gradient descent.
2. Evaluating the validation loss with a MLP forward pass $\mathcal{L}(\mathbf{w}; \mathcal{D}_{\text{val}})$.
3. Repeating the above steps for multiple epochs until either the maximum number of epochs has been reached or the validation loss has not improved for some chosen number of "patience" epochs, which is a standard early stopping criterion [13].

Furthermore, the final two layers of the MLP are frozen at initialisation to ensure asymmetry of the challenge instance for solution verification (see Section 6). The overall challenge structure is depicted in Figure 1, and the detailed structure of this standard training loop is given in Algorithm 1. The data generation, MLP construction and training loop components are deterministic conditioned on a random seed associated with the current challenge instance, which allows method verification reproducibility.

Goal The aim of the Innovators is to develop an optimizer such that when run in Algorithm 1, the MLPs MSE test error will be minimised

$$\mathcal{L}_{\text{MSE}}(\mathbf{w}; \mathcal{D}_{\text{test}}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \|y_i - \hat{f}_{\mathbf{w}}(\mathbf{x}_i)\|^2. \tag{3}$$

Submission constraints This challenge requires innovators to design components of a gradient descent iteration loop of the structure Algorithm 1 (with components like Algorithm 2 and Algorithm 3 provided by innovators) that can successfully optimise MLP parameters to perform regression on a dataset assessed by holdout test set performance. In particular, the optimiser step also has access to current MLP parameters, allowing for optimiser-inherent regularization techniques like weight decay [14]. In order to isolate contributions from optimiser innovation as much as possible, all training loop aspects outside of the optimiser algorithm (Algorithm 2) are preserved across all challenge instances. The backpropagation backbone for computing gradients in particular is identical across all challenge instances, as modifying this would result in moving away from gradient descent as in Direct Feedback Alignment [15].

Optimiser hyperparameters Note that innovators must specify not only the optimise algorithm, but also hyperparameters such as the learning rate. These hyperparameters should generally depend on challenge difficulty parameter, since gradient descent optimisers are known to be sensitive to hyperparameter choices [16]. Extended innovator rewards can be rewarded to improvements solely in hyperparameter selection for existing optimiser algorithm proposals.

4 Random Instance Generation

Dataset generation To model random functions, we choose GPs to model a distribution over functions, which provide a flexible infinite nonparametric family of functions while allowing fine control over analytical properties of the sample functions by choosing the kernel function $k(x, x')$ [17]. For generating infinitely differentiable $f(\cdot)$, we use RBF Kernels [18]. To do this in a memory efficient way we approximate the RBF

Algorithm 1 Standard training loop pseudocode that runs innovator submissions.

Input: training data $\mathcal{D}_{\text{train}} = \{\mathbf{x}_i, y_i\}_1^{N_{\text{train}}}$, validation data $\mathcal{D}_{\text{val}} = \{\mathbf{x}_i, y_i\}_1^{N_{\text{val}}}$, MLP architecture and initial parameters $\mathbf{w}_{\text{init}}$ split into a trainable $\mathbf{w}^{\text{tr}}$ and a frozen $\mathbf{w}^{\text{fr}}$ part, batch size B, patience P, difficulty parameter

Output: MLP parameters with the best validation loss $\mathbf{w}_*$, number of training epochs used T

```
Split $\mathcal{D}_{\text{train}}$ into $B$ equally sized batches $\mathcal{B}$
Set optimiser state $\mathcal{S} \leftarrow$ init_optimiser_state($\boldsymbol{\eta}$), $\mathbf{w} \leftarrow \mathbf{w}_{\text{init}}$, $\mathbf{g} \leftarrow \mathbf{0}$
Set previous average train and validation loss $\tilde{l}_{\text{train}} \leftarrow \text{NaN}$, $\tilde{l}_{\text{val}} \leftarrow \text{NaN}$
Set lowest validation loss observed $l_{\text{lowest}} \leftarrow \infty$, current epoch number $t \leftarrow 0$
while max fuel limit not reached do
  $l_{\text{train}} \leftarrow 0$
  for $\mathcal{D}_{\text{batch}}$ in shuffle_order($\mathcal{B}$) do
    $\tilde{\mathbf{w}}^{\text{tr}} \leftarrow$ query_location_parameters($\mathbf{w}^{\text{tr}}, \mathcal{S}, \mathbf{g}, t, \tilde{l}_{\text{train}}, \tilde{l}_{\text{val}}$) (Algorithm 3)
    $l, \mathbf{g} \leftarrow$ forward_and_backpropagation($\mathcal{L}_{\text{MSE}}([\tilde{\mathbf{w}}^{\text{tr}}, \mathbf{w}^{\text{fr}}]; \mathcal{D}_{\text{batch}})$)
    $\mathbf{u}, \mathcal{S} \leftarrow$ optimiser_step_updates($\mathcal{S}, \mathbf{g}, \mathbf{w}^{\text{tr}}, t, \tilde{l}_{\text{train}}, \tilde{l}_{\text{val}}$) (Algorithm 2)
    $\mathbf{w}^{\text{tr}} \leftarrow \mathbf{w}^{\text{tr}} + \mathbf{u}$ (gradient descent step)
    $l_{\text{train}} \leftarrow l_{\text{train}} + l/|\mathcal{B}|$
  end for
  $l_{\text{val}} \leftarrow \mathcal{L}_{\text{MSE}}(\mathbf{w}; \mathcal{D}_{\text{val}})$
  if $l_{\text{val}} < l_{\text{lowest}}$ then
    $\mathbf{w}_* \leftarrow \mathbf{w}$
    $l_{\text{lowest}} \leftarrow l_{\text{val}}$
  end if
  if validation loss has not improved for $P$ times in a row then
    Terminate loop at current epoch $T \leftarrow t + 1$ (early stopping)
  end if
  $\tilde{l}_{\text{train}} \leftarrow l_{\text{train}}$, $\tilde{l}_{\text{val}} \leftarrow l_{\text{val}}$
end while
return $\mathbf{w}_*, T$
```

Algorithm 2 Example pseudocode outline of vanilla stochastic gradient descent step with exponential learning rate decay as specified by innovator submissions.

Input: optimiser state $\mathcal{S}$, parameter gradients $\mathbf{g}$, trainable parameters $\mathbf{w}$, epoch number t, previous epoch training loss l_{train}, previous epoch validation loss l_{val}

Output: neural network parameter updates $\mathbf{u}$, new optimiser state $\mathcal{S}$

```
Get learning rate base and decay factor $\gamma_0, \tau_d \leftarrow \mathcal{S}$
Set learning rate $\gamma \leftarrow \gamma_0 \cdot e^{-t/\tau_d}$
for each parameter gradient $g_i$ in $\mathbf{g}$ do
  $u_i \leftarrow -\gamma \cdot g_i$
end for
return $\mathbf{u}, \mathcal{S}$ (state unchanged in this example)
```

Algorithm 3 Example pseudocode outline of "lookahead" gradients

Input: trainable parameters $\mathbf{w}$, current state $\mathcal{S}$, current gradient $\mathbf{g}$, epoch number t, previous epoch training loss l_{train}, previous epoch validation loss l_{val}

Output: location to compute gradient $\tilde{\mathbf{w}}$

```
$\tilde{\mathbf{w}} = \mathbf{w} + \mathbf{g}$
return $\tilde{\mathbf{w}}$
```

Kernel using Random Fourier Features. The data is generated as follows, sample the inputs $x_i \sim \text{Unif}[-1,-1]^D$, and set the targets y_i as

$$y_i = f(\mathbf{x}_i) + \xi_i, \quad f(\mathbf{x}) = \mathbf{a} \cdot \boldsymbol{\phi}(\mathbf{x}), \quad \mathbf{a} \sim \mathcal{N}(\mathbf{0}_K, \mathbf{I}_K), \tag{4}$$

where

$$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{\frac{2}{K}} \left[\cos\left(\boldsymbol{\omega}_1 \cdot \mathbf{x} + b_1\right), \ldots, \cos\left(\boldsymbol{\omega}_K \cdot \mathbf{x} + b_K\right)\right], \tag{5}$$

with $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}_D, l^{-2}\,\mathbf{I}_D)$ and $b \sim \text{Uniform}(0, 2\pi)$, where l is a parameter that governs the length scale of fluctuations in $f(\cdot)$. This is inspired by so Random Fourier Features for squared exponential Gaussian process kernels [19], where we limit K for scaling to large datasets.

For our challenge implementation, we pick a kernel lengthscale $l = 0.3$ across all $D = 2$ dimensions, and our number of basis functions as $K = 1000$. To prevent the benchmarker from anticipating what the test set is before running a complete training loop, we only use a subset of $\mathcal{D}_{\text{test}}$ where this subset is randomly selected based on a seed that depends on the number of training epochs T, which is not directly controlled by the optimiser algorithm due to early stopping.

Neural network generation Following standard literature, each network layer consists of a linear mapping, followed by nonlinear activation, and finally a batch normalisation layer [20]. The activation functions in hidden layers default to the commonly used rectified linear unit (ReLU) [21], which is robust to vanishing and exploding gradients [22, 23, 24] and have been empirically shown to outperform smooth activation functions such as the sigmoid [25] due to its tendency to create sparsity within the network as a regularising effect. We use standard Glorot initialisation of the MLP parameters [26] to obtain randomly initialised MLPs on which we apply the training loop. Different challenge instances have MLPs that vary in the number of hidden layers, or depth, while the layer widths are fixed at 256 for our challenge implementation.

5 Challenge Tracks

The tracks of the neural network challenge currently configure the size of the problem through the number of hidden layers (the depth). A larger depth requires more computation and could also lead to overfitting. All of the other parameters remain fixed over the tracks: size of the dataset, width, noise strength, etc.

6 Asymmetry Characteristics

It is clear that training a neural network is much more computationally expensive than running a single forward pass to obtain the test error loss.

Finding a function fit with good test error to the regression problem underlying this challenge is not inherently asymmetric (e.g. one can use kernel or Gaussian process regression [18]). However, the extremely high-dimensional search space for parameters $\mathbf{w}$ combined with the complex loss landscapes of neural networks makes it impractical to guess solutions or to use efficient solvers and convex optimisation techniques. We rely on this asymmetry of deep learning to keep the challenge asymmetric.

To avoid methods that can potentially sidestep this asymmetry for solution verification, like using previously trained neural networks to fine-tune them by training only final layers or performing linear regression on the final layer [1, 27], we freeze the final 2 layers at initialisation as shown in Figure 1A. This is because the required hidden layer input to the final two layers (which forms a smaller MLP subnetwork) is a highly complex problem in itself with a similar computational asymmetry to training MLPs,

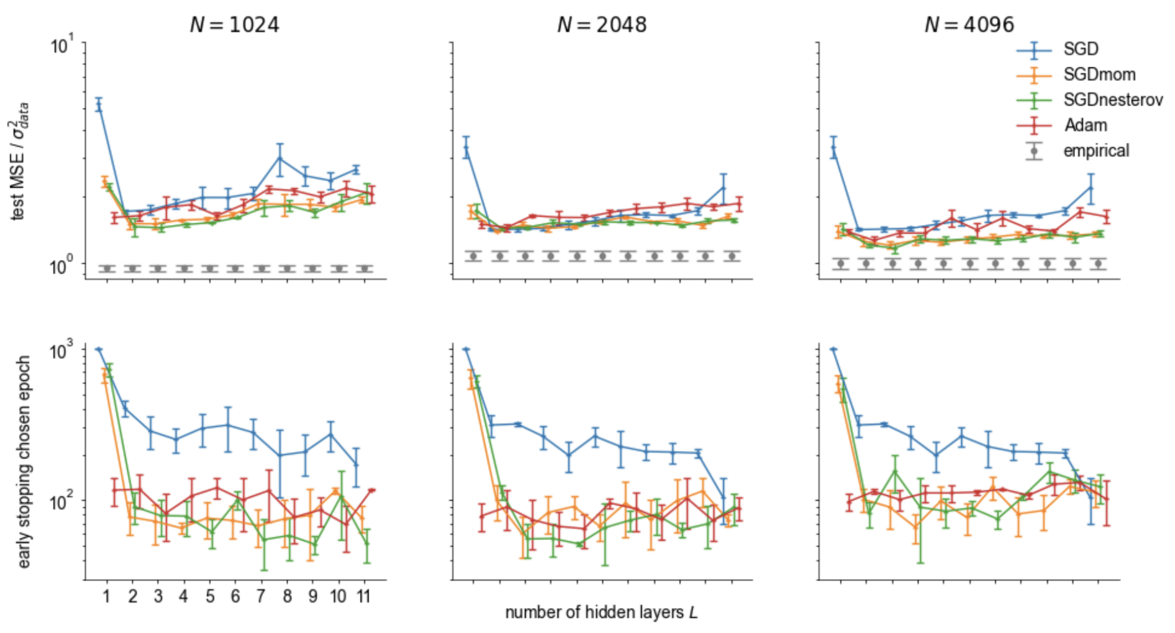

Figure 2: **Numerical exploration of MLP training results across challenge parameters.** Test mean squared error (top) and training epochs used before early stopping (bottom) across MLPs with varying numbers of hidden layers L, across three values for training loop dataset sizes N. Experiments are performed with vanilla SGD, moment variants of SGD and Adam. We also show in gray the empirical test set errors in the top panels. Error bars are s.e.m. computed over 3 random seed instances.

and thus forces any valid solution to require some procedure that matches the computational work of properly training MLPs.

Threshold Determination We require a baseline threshold to act as an anti-sybil mechanism. That is we cant allow benchmarkers to submit solutions of any quality since they could spam our validation systems. We determine a threshold at which the quality of a solution must be above for TIG to verify it. A simple approach to such a reference error is to use the empirical ground truth test mean squared error (which is a noisy estimate of σ^2_{data}) multiplied by some slack factor $\alpha > 1$

$$\epsilon^2_* = \frac{\alpha}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \|y_i - f_i\|^2. \tag{6}$$

We make an informed decision on the value(s) of α to pick for all challenge instances. We take the slack factor α to be independent of the difficulty parameters $\boldsymbol{\eta} = \{L, N\}$ with the following arguments:

- The space of accepted $\mathbf{w}$ does not depend on training data (only test data where the number of points is fixed).
- Increasing L (or generally network size) above a certain level does not meaningfully increase the challenge asymmetry as the dimensionality of $\mathbf{w}$ is already extremely high.
- Increasing N provides more potential signal to be extracted into the MLP parameters by the optimiser, but retains the asymmetry of finding valid $\mathbf{w}$ as the mapping from $\mathbf{w}$ to even a known target MLP function output remains highly nontrivial.

In Figure 2, we show numerical experiments of training MLP challenge instances for various difficulty parameter settings. We explore $L \in \{1, 2, \ldots, 11\}$ and $N = \{1024, 2048, 4096\}$ data points with a 80/20 train/validation split and a uniform hidden layer width of 256. For the optimiser algorithms, we apply vanilla SGD, its momentum variants and Adam optimisers with default hyperparameter values commonly used in the literature. We can observe the following:

- In top panels, we observe that deeper MLPs are harder to train and require more advanced versions of SGD to reach better test errors, with an optimal depth $L_* \approx 3$ for the dataset type used.
- In bottom panels, we visualize the number of epochs used before early stopping and observe more efficient training trajectories for the advanced versions of SGD.
- Consistent with the literature, Adam does not lead to the best generalisation performance, with SGD momentum variants outperforming in test error.
- Adam outperforms all SGD variants for $L = 1$, where the hidden layer is frozen (it is part of the final two layers including the output layer) and we thus observe training an effectively linear model (input layer) with a complicated non-quadratic effective loss function due to the final frozen MLP subnetwork.

Across all difficulty parameters explored, we observe a slack factor of $\alpha = 4$ to provide a good margin for qualifying MLP solutions for $L > 2$, ignoring $L = 1$ since it is an effective linear model edge case. Note that due to the nonconvex nature of neural network training, numerical instabilities or non-convergence may generally occur during the training iterations for arbitrarily designed optimisers.

7 Future Outlook

There are many avenues to pursue for extending this challenge to have more potential real-world impact, for example:

- Include different types of neural network architectures like transformers [3] and recurrent neural networks [23]
- Consider the variation of training loop hyperparameters like batch size and early stopping patience, which is known to affect training results significantly [12]
- Allow a more general optimisation loop to be modified by the innovators beyond the gradient descent step alone
- Consider subtasks which focus on optimizing particular classes of neural networks/types of datasets, e.g. LLMs and text or RNNs and time series

We anticipate to gain valuable feedback from the community and are open to modifying design details necessary for aligning this challenge with the target goals set out in Section 1.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[4] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[5] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[8] Diederik Kinga, Jimmy Ba Adam, et al. A method for stochastic optimization. In *International conference on learning representations (ICLR)*, volume 5. California;, 2015.

[9] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. Place: US Publisher: American Psychological Association.

[10] Anna Choromanska, Yann LeCun, and Gérard Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pages 1756–1760. PMLR, 2015.

[11] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In *International conference on machine learning*, pages 2603–2612. PMLR, 2017.

[12] Chaoyue Liu and Mikhail Belkin. Accelerating sgd with momentum for over-parameterized learning. *arXiv preprint arXiv:1810.13395*, 2018.

[13] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[14] Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

[15] Julien Launay, Iacopo Poli, François Boniface, and Florent Krzakala. Direct feedback alignment scales to modern deep learning tasks and architectures. *Advances in neural information processing systems*, 33:9346–9360, 2020.

[16] Rhian Taylor, Varun Ojha, Ivan Martino, and Giuseppe Nicosia. Sensitivity analysis for deep learning: ranking hyper-parameter influence. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 512–516. IEEE, 2021.

[17] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, June 2008. Publisher: Institute of Mathematical Statistics.

[18] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, Cambridge, MA, 2006. OCLC: ocm61285753.

[19] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[20] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[21] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, June 2000. Publisher: Nature Publishing Group.

[22] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. Conference Name: IEEE Transactions on Neural Networks.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, February 2013. arXiv:1211.5063.

[24] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A Convergence Theory for Deep Learning via Over-Parameterization, June 2019. arXiv:1811.03962.

[25] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, June 2011. ISSN: 1938-7228.

[26] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. ISSN: 1938-7228.

[27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.