Learning by Turning: Neural Architecture Aware Optimisation

Yang Liu [* 1]   Jeremy Bernstein [* 2]   Markus Meister [2]   Yisong Yue [2]

Abstract

Descent methods for deep networks are notoriously capricious: they require careful tuning of step size, momentum and weight decay, and which method will work best on a new benchmark is a priori unclear. To address this problem, this paper conducts a combined study of neural architecture and optimisation, leading to a new optimiser called *Nero*: the neuronal rotator. Nero trains reliably without momentum or weight decay, works in situations where Adam and SGD fail, and requires little to no learning rate tuning. Also, Nero's memory footprint is $\sim$ square root that of Adam or LAMB. Nero combines two ideas: (1) projected gradient descent over the space of *balanced networks*; (2) neuron-specific updates, where the step size sets the *angle* through which each neuron's hyperplane *turns*. The paper concludes by discussing how this geometric connection between architecture and optimisation may impact theories of generalisation in deep learning.

1. Introduction

Deep learning has brought on a new paradigm in computer science, enabling artificial systems to interact with the world at an unprecedented level of complexity. That said, the core technology relies on various heuristic numerical techniques that are sometimes brittle and often require extensive tuning. A major goal of modern research in machine learning is to uncover the principles underlying learning in neural systems, and thus to derive more reliable learning algorithms.

Part of the challenge of this endeavour is that learning in deep networks is an inherently coupled problem. Suppose that training performance is sensitive to a particular detail of the neural architecture—then it is unclear whether that detail affects the expressivity of the architecture, or just the ability of the descent method to train the architecture.

---

[*]Equal contribution [1]Abacus.AI [2]Caltech. Correspondence to: YL <yang@abacus.ai> and JB <bernstein@caltech.edu>. Code available at `github.com/jxbz/nero`.

This observation motivates the *combined study* of architecture and optimisation, and this paper explores several questions at that intersection. First of all:

⟨?⟩ What is the right domain of optimisation for a neural network's weights? Is it $\mathbb{R}^d$, or something more exotic—such as a Cartesian product of hyperspheres?

Typically, optimisation is conducted over $\mathbb{R}^d$, while a careful weight initialisation and a tuned weight decay hyperparameter impose a soft constraint on the optimisation domain. Since normalisation schemes such as batch norm (Ioffe & Szegedy, 2015) render the network invariant to the scale of the weights, weight decay also plays a somewhat subtle second role in modifying the effective learning rate. Hyperparameters with this kind of subtle coupling add to the compounding cost of hyperparameter search.

Furthermore, descent methods such as Adam (Kingma & Ba, 2015) and LAMB (You et al., 2020) use either synapse-specific or layer-specific gradient normalisation. This motivates a second question:

⟨?⟩ At what level of granularity should an optimiser work? Should normalisation occur per-synapse or per-layer—or perhaps, per-neuron?

This paper contends that in deep learning, hyperparameters proliferate because of hidden couplings between optimiser and architecture. By studying the above questions, and distilling simple rules that govern optimisation and architecture, this paper aims to make deep learning less brittle—and less sensitive to opaque hyperparameters.

Summary of contributions:

1. A new optimiser—*Nero*: the neuronal rotator. Nero performs per-neuron projected gradient descent, and uses $\sim$ square root the memory of Adam or LAMB.

2. Experiments across image classification, image generation, natural language processing and reinforcement learning, in which Nero's *out-of-the-box* configuration tends to outperform *tuned* baseline optimisers.

3. Discussion of how the connection between optimisation and architecture relates to generalisation theories, such as PAC-Bayes and norm-based complexity.

2. Related work

This section reviews relevant work pertaining to both neural architecture design and optimisation in machine learning, and concludes with a bridge to the neuroscience literature.

2.1. Neural Architecture Design

The importance of wiring constraints for the stable function of engineered neural systems is not a new discovery. One important concept is that of *balanced excitation and inhibition*. For instance, Rosenblatt (1958) found that balancing the proportion of excitatory and inhibitory synaptic connections made his perceptron more robust to varying input sizes. Another concept relates to the *total magnitude of synapse strengths*. For example, Rochester et al. (1956) constrained the sum of magnitudes of synapses impinging on a neuron so as to stabilise the process of learning. Similar ideas were explored by von der Malsburg (1973) and Miller & MacKay (1994). These works are early predecessors to this paper's definition of *balanced networks* given in Section 3.1.

Given the resurgence of neural networks over the last decade, the machine learning community has taken up the mantle of research on neural architecture design. Special weight scalings—such as *Xavier init* (Glorot & Bengio, 2010) and *Kaiming init* (He et al., 2015)—have been proposed to stabilise signal transmission through deep networks. These scalings are only imposed at initialisation and are free to wander during training—an issue which may be addressed by tuning a weight decay hyperparameter. More recent approaches—such as batch norm (Ioffe & Szegedy, 2015)—explicitly control activation statistics throughout training by adding extra normalisation layers to the network.

Other recent normalisation techniques lie closer to the work of Rosenblatt (1958) and Rochester et al. (1956). Techniques that involve constraining a neuron's weights to the unit hypersphere include: weight norm (Salimans & Kingma, 2016), decoupled networks (Liu et al., 2017; 2018) and orthogonal parameterised training (Liu et al., 2021). Techniques that also balance excitation and inhibition include centred weight norm (Huang et al., 2017) and weight standardisation (Qiao et al., 2019).

2.2. Descent Methods in Deep Learning

Much classic work in optimisation theory focuses on deriving convergence results for descent methods under assumptions such as *convexity* (Boyd & Vandenberghe, 2004) and *Lipschitz continuity of the gradient* (Nesterov, 2004). These simplifying assumptions are often used in the machine learning literature. For instance, Bottou et al. (2018) provide convergence guarantees for stochastic gradient descent (SGD) under each of these assumptions. However, these assumptions do not hold in deep learning (Sun, 2019).

On a related note, SGD is not the algorithm of choice in many deep learning applications, and heuristic methods such as RMSprop (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2015) often work better. For instance, Adam often works much better than SGD for training generative adversarial networks (Bernstein et al., 2020a). Yet the theory behind Adam is poorly understood (Reddi et al., 2018).

A more recent line of work has explored optimisation methods that make *relative updates* to neural network parameters. Optimisers like LARS (You et al., 2017), LAMB (You et al., 2020) and Fromage (Bernstein et al., 2020a) make per-layer relative updates, while Madam (Bernstein et al., 2020b) makes per-synapse relative updates. You et al. (2017) found that these methods stabilise large batch training, while Bernstein et al. (2020a) found that they require little to no learning rate tuning across tasks.

Though these recent methods partially account for the neural architecture—by paying attention to its layered operator structure—they do not rigorously address the optimisation domain. As such, LARS and LAMB require a tunable weight decay hyperparameter, while Fromage and Madam restrict the optimisation to a bounded set of tunable size (i.e. weight clipping). Without this additional tuning, these methods can be unstable—see for instance (Bernstein et al., 2020a, Figure 2) and (Bernstein et al., 2020b, Figure 3).

The discussion in the previous paragraph typifies the machine learning state of the art: optimisation techniques that work well, albeit only after hyperparameter tuning. For instance, LAMB is arguably the state-of-the-art relative optimiser, but it contains in total *five* tunable hyperparameters. Since—at least naïvely—the cost of hyperparameter search is exponential in the number of hyperparameters, the prospect of fully tuning LAMB is computationally daunting.

2.3. Homeostatic Control in Neuroscience

Since the brain is a system that must learn stably without hyperparameter do-overs, it is worth looking to neuroscience for inspiration on designing better learning algorithms.

A major swathe of neuroscience research studies mechanisms by which the brain performs homeostatic control. For instance, neuroscientists report a form of homeostasis termed *synaptic scaling*, where a neuron modulates the strengths of all its synapses to stabilise its firing rate (Turrigiano, 2008). More generally, *heterosynaptic plasticity* refers to homeostatic mechanisms that modulate the strength of unstimulated synapses (Chistiakova et al., 2015). Shen et al. (2020) review connections to normalisation methods used in machine learning.

These observations inspired this paper to consider implementing homeostatic control via projected gradient descent—leading to the Nero optimiser.

3. Background Theory

In general, an L-layer neural network $f(\cdot)$ is a composition of L simpler functions $f_1(\cdot), ..., f_L(\cdot)$:

$$f(x) = f_L \circ f_{L-1} \circ ... \circ f_1(x). \qquad \text{(forward pass)}$$

Due to this compositionality, any slight ill-conditioning in the simple functions $f_i(\cdot)$ has the potential to *compound* over layers, making the overall network $f(\cdot)$ very ill-conditioned. Architecture design should aim to prevent this from happening, as will be covered in Section 3.1.

The Jacobian $\partial f / \partial f_l$, which plays a key role in evaluating gradients, also takes the form of a deep product:

$$\frac{\partial f}{\partial f_l} = \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial f_{L-2}} \cdot ... \cdot \frac{\partial f_{l+1}}{\partial f_l}. \quad \text{(backward pass)}$$

Therefore, it is also important from the perspective of gradient-based optimisation that compositionality is adequately addressed, as will be covered in Section 3.2.

3.1. Balanced Network Architectures

A common strategy to mitigate the issue of compounding ill-conditioning is to explicitly re-normalise the activations at every network layer. Batch norm (Ioffe & Szegedy, 2015) exemplifies this strategy, and was found to improve the trainability of deep residual networks. Batch norm works by standardising the activations across a batch of inputs at each network layer—that is, it shifts and scales the activations to have mean zero and variance one across a batch.

Although batch norm works well, it adds computational overhead to both the forward and backward pass. To explore how far one can get without explicit re-normalisation, the following definitions are useful:

Definition 1. A neuron is *balanced* if its weight vector $w \in \mathbb{R}^d$ satisfies the following constraints:

$$\sum_{i=1}^d w_i = 0; \qquad \text{(balanced excitation \& inhibition)}$$

$$\sum_{i=1}^d w_i^2 = 1. \qquad (\ell_2 \text{ constant sum rule})$$

Definition 2. A network is *balanced* if all its constituent neurons are balanced.

As noted by Huang et al. (2017), balanced neurons attain some of the properties of batch norm for free. To see this, consider a linear neuron $y = \sum_i w_i x_i$ with inputs x_i that are uncorrelated with mean μ and variance 1. Then the output y is standardised:

$$\mathbb{E}[y] = \sum_i w_i \, \mathbb{E}[x_i] = \mu \sum_i w_i = 0;$$
$$\text{Var}[y] = \sum_i w_i^2 \, \text{Var}[x_i] = \sum_i w_i^2 = 1.$$

While the assumptions on the inputs x_i are unlikely to hold exactly, under more general conditions the constraints may at least *encourage* the standardisation of activation statistics through the layers of the network (Brock et al., 2021).

3.2. Stable Descent Steps

Since a network is trained via perturbations to its parameters, it is important to know what size perturbations are appropriate. Consider an L-layer network with weight matrices $W = (W_1, W_2, ..., W_L)$ and loss function $\mathcal{L}(W)$. For a perturbation $\Delta W = (\Delta W_1, \Delta W_2, ..., \Delta W_L)$, the following definition establishes a notion of stable step size:

Definition 3. Let θ_l denote the angle between ΔW_l and $-\nabla_{W_l} \mathcal{L}(W)$. A descent step is *stable* if for all $l = 1, ..., L$:

$$\frac{\|\nabla_{W_l} \mathcal{L}(W + \Delta W) - \nabla_{W_l} \mathcal{L}(W)\|_F}{\|\nabla_{W_l} \mathcal{L}(W)\|_F} < \cos \theta_l. \qquad (1)$$

Or in words: for each layer, the relative change in gradient induced by the perturbation should not exceed the cosine of the angle between the perturbation and the negative gradient.

This definition is useful because a stable descent step is guaranteed to decrease a continuously differentiable loss function $\mathcal{L}(W)$ (Bernstein et al., 2020a). Still, extracting a stable step ΔW directly from Inequality 1 would require first computing extra gradients $\nabla_{W_l} \mathcal{L}(W + \Delta W)$. Bernstein et al. (2020a) proposed the following model to avoid this:

Definition 4. The loss function obeys *deep relative trust* if for all perturbations $\Delta W = (\Delta W_1, \Delta W_2, ..., \Delta W_L)$:

$$\frac{\|\nabla_{W_l} \mathcal{L}(W + \Delta W) - \nabla_{W_l} \mathcal{L}(W)\|_F}{\|\nabla_{W_l} \mathcal{L}(W)\|_F} \leqslant \prod_{k=1}^L \left(1 + \frac{\|\Delta W_k\|_F}{\|W_k\|_F}\right) - 1.$$

While deep relative trust is based on a perturbation analysis of L-layer perceptrons (Bernstein et al., 2020a, Theorem 1), the key idea is that its product structure explicitly models the product structure of the network's backward pass.

The deep relative trust model suggests that a stable descent step should involve small relative perturbations *per layer*. This motivates the layer-wise family of descent methods (You et al., 2017; 2020). Still, it is unclear whether layers are the right base object to consider. Perhaps a more refined analysis would replace the layers appearing in Definition 4 with individual *neurons* or even *synapses*.

Small relative perturbations per-synapse were explored by Bernstein et al. (2020b) and found to slightly degrade training performance compared to Adam and SGD. But this paper will explore the per-neuron middle ground:

Definition 5. A step of size $\eta > 0$ is said to be *per-neuron relative* if for any neuron with weights $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$, the perturbations $\Delta w \in \mathbb{R}^d$ and $\Delta b \in \mathbb{R}$ satisfy:

$$\|\Delta w\|_2 / \|w\|_2 \leqslant \eta \qquad \text{and} \qquad |\Delta b|/|b| \leqslant \eta.$$

A per-neuron relative update is automatically per-layer relative. To see this, consider a weight matrix W whose N rows correspond to N neurons $w^{(1)}, ..., w^{(N)}$. Then:

$$\frac{\|\Delta W\|_F}{\|W\|_F} = \sqrt{\frac{\sum_{i=1}^N \|\Delta w^{(i)}\|_2^2}{\sum_{i=1}^N \|w^{(i)}\|_2^2}} \leqslant \sqrt{\frac{\sum_{i=1}^N \eta^2 \|w^{(i)}\|_2^2}{\sum_{i=1}^N \|w^{(i)}\|_2^2}} = \eta. \quad (2)$$

4. Nero: the <u>Ne</u>uronal <u>Ro</u>tator

Following the discussion in Section 3, this paper will consider an optimisation algorithm that makes *per-neuron relative updates* (Definition 5) constrained to the space of *balanced networks* (Definition 2).

Since a balanced neuron is constrained to the unit hypersphere, a per-neuron relative update with step size η corresponds to a pure rotation of the neuron's weight vector by angle $\approx \eta$. To see this, take η small in the following picture:

$$\|w + \Delta w\|_2 = 1$$
$$\theta \qquad \|\Delta w\|_2 = \eta$$
$$\|w\|_2 = 1$$
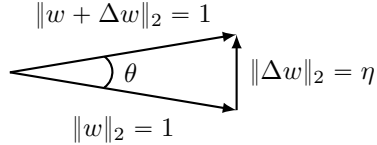
Hence, this paper proposes *Nero*: the <u>ne</u>uronal <u>ro</u>tator. Nero's goal is to reduce the burden of hyperparameter tuning by baking architectural information into the optimiser. More concretely, the anticipated advantages are as follows:

1. Since per-neuron relative updates are automatically per-layer relative by Equation 2, they should inherit the properties of per-layer updates—in particular, stability across batch sizes (You et al., 2017) while needing little to no learning rate tuning (Bernstein et al., 2020a).

2. Since balanced networks place hard constraints on the norm of a neuron's weights, the need for initialisation tuning and weight decay on these weights is removed.

3. Gradients are often normalised by running averages, in order to retain relative scale information between successive minibatch gradients (Tieleman & Hinton, 2012). Along with momentum, this is the main memory overhead of Adam and LAMB compared to vanilla SGD. Per-neuron running averages consume $\sim$ square root the memory of per-synapse running averages.

4. Since normalisation is local to a neuron, no communication is needed between neurons in a layer (unlike for per-layer updates). This makes the optimiser more distributable—for example, a single layer can be split across multiple compute devices without fuss. For the same reason, the Nero update seems more biologically plausible than per-layer optimisers such as LAMB.

There is a significant difference between the implementation of balanced networks in Nero versus prior work. In centred weight norm (Huang et al., 2017) and weight standardisation (Qiao et al., 2019), a neuron's underlying weight representation is an *unnormalised* vector $\widetilde{w} \in \mathbb{R}^d$—which is normalised by including the following reparameterisation in the neural architecture:

$$\texttt{normalise}(\widetilde{w}) := \frac{\widetilde{w} - \mathbf{1}^T \widetilde{w} \cdot \mathbf{1}/d}{\|\widetilde{w} - \mathbf{1}^T \widetilde{w} \cdot \mathbf{1}/d\|_2}, \qquad (3)$$

where $\mathbf{1}$ denotes the vector of 1s.

Algorithm 1 Nero optimiser. "Out-of-the-box" hyperparameter defaults are $\eta = 0.01$ and $\beta = 0.999$. The constant $\sigma_b \in \mathbb{R}_+$ refers to the initialisation scale of the biases.

Input: step size $\eta \in (0, 1]$, averaging constant $\beta \in [0, 1)$
repeat
  for each neuron do
    $\triangleright$ get weight & bias gradients $g_w \in \mathbb{R}^n$ & $g_b \in \mathbb{R}$
    $\triangleright$ update running averages
    $\bar{g}_w^2 \leftarrow \beta \cdot \bar{g}_w^2 + (1 - \beta) \cdot \|g_w\|_2^2$
    $\bar{g}_b^2 \leftarrow \beta \cdot \bar{g}_b^2 + (1 - \beta) \cdot g_b^2$
    $\triangleright$ update weights $w \in \mathbb{R}^n$ and bias $b \in \mathbb{R}$
    $w \leftarrow w - \eta \cdot \|w\|_2 / \bar{g}_w \cdot g_w$
    $b \leftarrow b - \eta \cdot \sigma_b / \bar{g}_b \cdot g_b$
    $\triangleright$ project weights back to constraint set
    $w \leftarrow w - \frac{1}{n} \sum_{i=1}^n w_i$
    $w \leftarrow w / \|w\|_2$
  end for
until converged

Since the target of automatic differentiation is still the unnormalised vector $\widetilde{w}$, overhead is incurred in both the forward and backward pass. Moreover, there is a subtle coupling between the step size in additive optimisers like Adam and the scale of the unnormalised weights $\widetilde{w}$—see Section 5.3.

In contrast, Nero opts to implement balanced networks via projected gradient descent. This is lighter-weight than Equation 3, since duplicate copies of the weights are not needed and the network's backward pass does not involve extra operations. Furthermore, Nero can be used as a drop-in replacement for optimisers like Adam, SGD or LAMB, without the user needing to manually modify the network architecture via the reparameterisation in Equation 3. Note that projected gradient descent arises frequently in machine learning (Chen et al., 2019; Bai et al., 2019).

Pseudocode for Nero is provided in Algorithm 1. Since Nero normalises gradients via running averages, a Nero update is only approximately per-neuron relative. For brevity, the Adam-style bias correction of the running averages is omitted from the pseudocode. But in the Pytorch implementation used in this paper's experiments, the running averages $\bar{g}_w$ and $\bar{g}_b$ are divided by a factor of $\sqrt{1 - \beta^t}$ before the tth update. This corrects for the warmup bias stemming from $\bar{g}_w$ and $\bar{g}_b$ being initialised to zero (Kingma & Ba, 2015).

While the pseudocode in Algorithm 1 is presented for *neurons* and *biases*, in the Pytorch implementation the bias update is applied to any parameters lacking a notion of fan-in—including batch norm gains and biases. Typical initialisation scales are $\sigma_b = 1$ for gains and $\sigma_b = 0.01$ for biases. The Pytorch implementation of Nero defaults to $\sigma_b = 0.01$ for any bias parameter initialised to zero.

5. Experiments

This section presents experiments intended to demonstrate Nero's key properties. In all figures, the mean and range are plotted over three repeats. For Nero, *out-of-the-box* refers to setting $\eta = 0.01$ and $\beta = 0.999$. The code for these experiments is available at `github.com/jxbz/nero`, and more experimental details are given in Appendix A.

5.1. Constraints Help Nero

To verify that projecting to the space of balanced networks improves the performance of Nero, an ablation experiment was conducted. As can be seen in Figure 1, when training a VGG-11 image classifier on the CIFAR-10 dataset, Nero performed best with both constraints switched on.

5.2. Per-Neuron Updates are a Good Middle Ground

Since Bernstein et al. (2020b) found that per-synapse relative updates led to slightly degraded performance, while per-layer relative updates typically perform well (You et al., 2017; 2020; Bernstein et al., 2020a), this section compares per-synapse, per-neuron and per-layer relative updates. In particular, Nero is compared to Madam (per-synapse relative) and LAMB (per-layer relative).

A VGG-11 model was trained on the CIFAR-10 dataset. Without constraints, the three optimisers performed similarly, achieving $\sim 12\%$ top-1 validation error (Figure 2, top). Constraining to the space of balanced networks (Definition 2) improved both Nero and LAMB, but did not have a significant effect on Madam (Figure 2, bottom). In both configurations, Nero outperformed Madam and LAMB, demonstrating the viability of per-neuron relative updates.

5.3. The Pitfalls of Reparameterisation

Existing implementations of balanced networks (Definition 2) work via the re-parameterisation given in Equation 3 (Huang et al., 2017; Qiao et al., 2019). This leads to an undesired coupling between the learning rate in optimisers like Adam and the scale of the unnormalised $\widetilde{w}$ parameters.

To verify this, a network with weights normalised by Equation 3 was trained to classify the MNIST dataset. The initial weights $\widetilde{w}$ were drawn from $\mathcal{N}(0, \sigma^2)$, and the experiment was repeated for $\sigma = 1$ and $\sigma = 100$. The Adam optimiser was used for training with a fixed learning rate of 0.01. As can be seen in Figure 3 (left), the training performance was sensitive to the weight scale σ, despite the fact that a weight normalisation scheme was being used.

The unnecessary scale freedom of reparameterisation can lead to other undesired consequences such as numerical overflow. Nero completely eliminates this issue by implementing balanced networks via projected gradient descent.
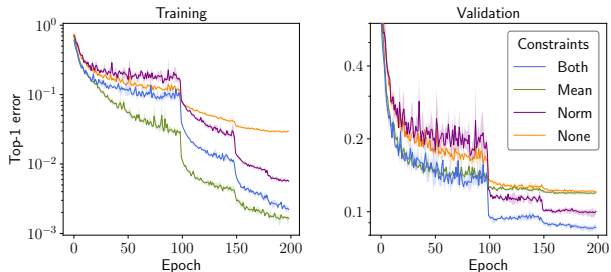

Figure 1. Ablating the *balanced network* constraints. A VGG-11 network was trained on CIFAR-10. The legend denotes which of Nero's constraints were active. *Mean* refers to balanced excitation & inhibition, while *norm* refers to the ℓ_2 constant sum rule.
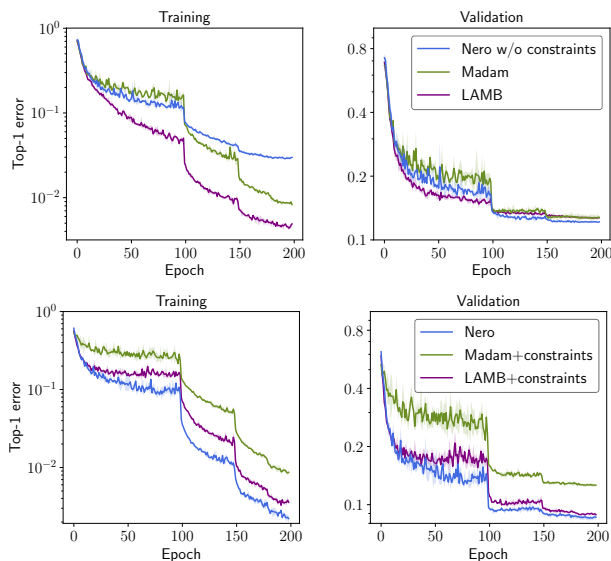

Figure 2. Comparing per-synapse (Madam), per-neuron (Nero) and per-layer (LAMB) relative updates. A VGG-11 network was trained to classify CIFAR-10. Top: all optimisers *without* balanced network constraints. Bottom: all optimisers *with* constraints.
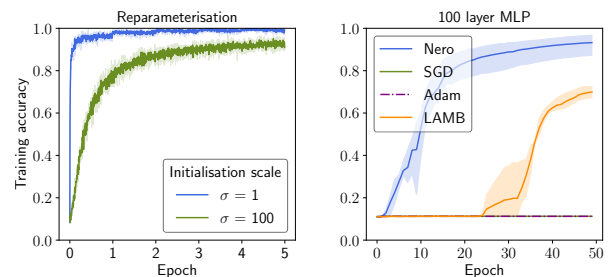

Figure 3. Left: Training a 5 layer perceptron normalised via reparameterisation (Equation 3) on MNIST. For a fixed Adam learning rate, training is sensitive to the scale σ of the raw weights $\widetilde{w}$. This motivates the different approach taken by Nero. Right: Using Nero to train a 100 layer perceptron—without batch norm or skip connections—to classify MNIST.

5.4. Nero Trains Deeper Networks

Very deep networks are typically difficult to train without architectural modifications such as residual connections (He et al., 2016) or batch norm (Ioffe & Szegedy, 2015). To test whether Nero enables training very deep models without such modifications, Figure 3 (right) shows the results of training a very deep multilayer perceptron (MLP) on the MNIST dataset. Unlike SGD, Adam and LAMB, Nero could reliably train a 100-layer MLP.

5.5. Nero Works Well Out-of-the-Box

This section probes the versatility and robustness of Nero by comparing its optimisation and generalisation performance with three popular alternatives—SGD, Adam, and LAMB—across six learning problems. The tasks span the domains of computer vision, natural language processing, and reinforcement learning. A wide spectrum of neural architectures were tested—from convolutional networks to transformers.

To make a fair comparison between optimisers, a fair hyperparameter tuning strategy is needed. In this section:

1. Learning rates were tuned over $\{10^{-4}, 10^{-3}, ..., 10^0\}$.

2. For Adam, LAMB and SGD, the momentum hyperparameter was tuned to achieve good performance on the most complicated benchmark—cGAN training—and then fixed across the rest of the benchmarks. In each case, the best momentum value for cGAN was 0.

3. β in Nero and β_2 in Adam and LAMB were fixed to 0.999 across all experiments, as recommended by Kingma & Ba (2015) and You et al. (2020).

4. Weight decay was not used in any of the experiments.

The results are collated in Table 1. Nero achieved the best validation performance in every experiment—while the runner-up varied across tasks. What's more, the same learning rate of $\eta = 0.01$ was optimal for Nero in five out of six experiments. This means that Nero has strong *out-of-the-box* performance, since Nero's only other hyperparameter was fixed to $\beta = 0.999$ across all experiments.

The remainder of this section discusses each experiment in turn. Implementation details are given in Appendix A.

Image synthesis with cGAN Generative Adversarial Network (Goodfellow et al., 2014, GAN) training is perhaps the most challenging optimisation problem tackled in this paper. Good performance has traditionally relied on extensive tuning: different learning rates are often used in the generator and discriminator (Heusel et al., 2017) and training is highly sensitive to momentum (Brock et al., 2019, p. 35). The class-conditional GAN model in this paper is based on the BigGAN architecture (Brock et al., 2019). This is a heterogeneous network involving a variety of building
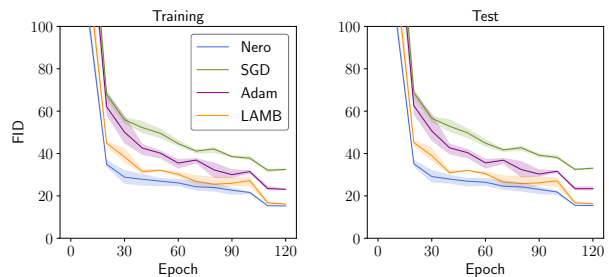

Figure 4. Class-conditional GAN training on CIFAR-10. Equal learning rates were used in the generator and discriminator. The Fréchet Inception Distance (Heusel et al., 2017, FID) measures the distance between the sample statistics of real and fake data as represented at a deep layer of a pre-trained image classifier.
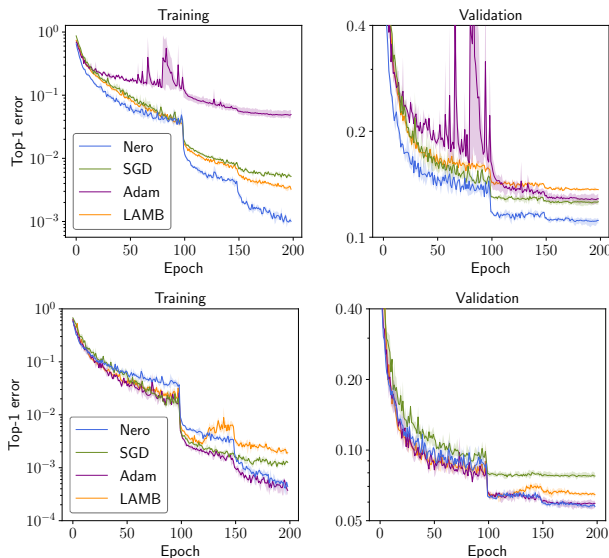

Figure 5. CIFAR-10 classification. Top: performance of a vanilla, convolutional VGG-11 network. Bottom: performance of a batch-normalised, residual ResNet-18 network.
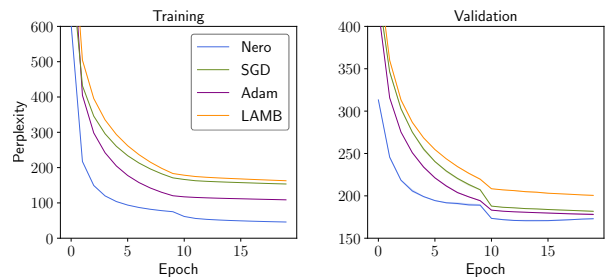

Figure 6. Training a language model on the Wikitext-2 dataset. A small transformer network was used, composed of 19 tensors. Nero achieved the best anytime performance.

Task	Dataset	Model	Metric (↑↓)	Nero	SGD	Adam	LAMB	Nero η	SGD η	Adam η	LAMB η
cGAN	CIFAR-10	BigGAN-like	FID (↓)	**15.43 ± 0.37**	33.06 ± 0.42	23.42 ± 0.85	<u>16.32 ± 0.23</u>	0.01	0.01	0.0001	0.01
Classification	CIFAR-10	VGG11	Top-1 Error (↓)	**11.16% ± 0.17**	<u>12.61% ± 0.21</u>	12.86% ± 0.34	13.66% ± 0.05	0.01	0.1	0.001	0.01
Classification	CIFAR-10	ResNet-18	Top-1 Error (↓)	**5.75% ± 0.07**	7.75% ± 0.17	<u>5.93% ± 0.19</u>	6.46% ± 0.12	0.01	0.1	0.01	0.1
Language Model	Wikitext-2	Transformer	Perplexity (↓)	**172.99 ± 0.51**	181.76 ± 0.49	<u>178.05 ± 0.96</u>	200.54 ± 0.53	0.01	1.0	0.0001	0.01
Translation	WMT16 En–De	Transformer	Perplexity (↓)	**11.35 ± 1.20**	92.40 ± 89.48	<u>12.63 ± 0.34</u>	16.36 ± 0.29	0.001	0.0001	0.0001	0.01
PPO	Atari Pong	vanilla CNN	Reward (↑)	**20.62 ± 0.05**	11.99 ± 8.65	<u>15.92 ± 3.40</u>	−19.46 ± 0.10	0.01	0.1	0.0001	0.001

Table 1. Validation results for the best learning rate η. The best result is shown in bold, while the runner-up is underlined.

blocks: convolutions, embeddings, fully connected layers, attention layers, conditional batch norm and spectral norm (Miyato et al., 2018). The results are presented in Figure 4.

Image classification Experiments were run across all baselines on the CIFAR-10 dataset. The networks used were the vanilla, convolutional VGG-11 network (Simonyan & Zisserman, 2015) and the batch-normalised, residual ResNet-18 network (He et al., 2015). The results are presented in Figure 5. ImageNet results using ResNet-50 are presented in Section 5.6. Due to limited computational resources, the LAMB and Adam baselines were omitted.

Natural language processing Much recent progress in natural language processing is based on the transformer architecture (Vaswani et al., 2017). Transformers process information via layered, all-to-all comparisons—without recourse to recurrence or convolution. This paper experimented with a smaller transformer (19 tensors) trained on the Wikitext-2 dataset, and a larger transformer (121 tensors) trained on WMT2016 English–German translation. The results are presented in Figures 6 and 7.

Reinforcement learning Many reinforcement learning algorithms use neural networks to perform function approximation. Proximal Policy Optimization (Schulman et al., 2017, PPO) is one example, and PPO has gained increasing popularity for its simplicity, scalability, and robust performance. This paper experimented with PPO on the Atari Pong video game. The results are presented in Figure 8.

While LAMB failed to train on this task, further investigation revealed that setting LAMB's momentum hyperparameter to 0.9 enabled LAMB to learn. This demonstrates that LAMB is sensitive to the momentum hyperparameter.

5.6. Nero Can Be Regularised

This section compares using Nero versus SGD to train a ResNet-50 classifier on the ImageNet dataset. The results are shown in Figure 9. While out-of-the-box Nero attained the best training error and better validation error than SGD, it performed worse than SGD with tuned weight decay on the validation set. But after fine-tuning the learning rate and adding regularisation, Nero roughly matched SGD with weight decay. In particular, the tuned version of Nero used a learning rate of 0.02 (tuned), a bias scale parameter $\sigma_b = 1.0$ (not tuned) and the batch norm gains were regularised towards one using a quadratic penalty.
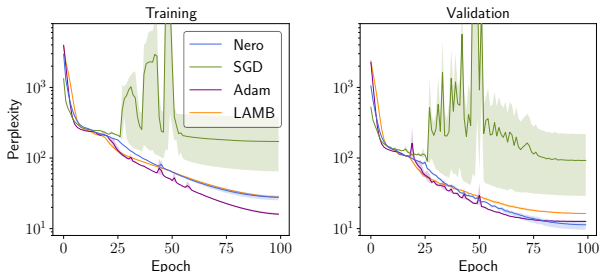

Figure 7. Training an English–German translation model on WMT16. A larger transformer network was used, composed of 121 tensors. The optimisers with gradient normalisation—Nero, Adam, and LAMB—performed best in training this model. Training with SGD was unstable and led to significantly worse perplexity.
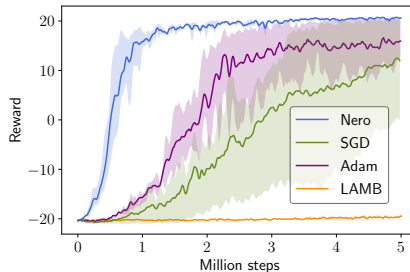

Figure 8. Training a policy network to play Pong. Proximal Policy Optimisation (PPO) was used. Pong's reward is bounded between ± 21. While investigating LAMB's failure to train the policy network, it was discovered that adjusting the β_1 momentum hyperparameter from 0 to 0.9 improved LAMB's performance.
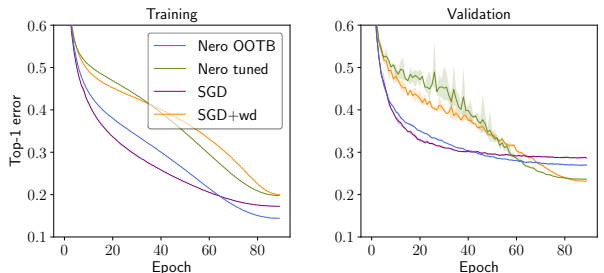

Figure 9. Training a ResNet-50 network to classify the ImageNet dataset. *Nero OOTB* (out-of-the-box) achieved the best training performance but overfit compared to SGD with weight decay. *Nero tuned*—which most importantly regularised batch norm gains towards one—recovered most of the lost performance.

6. Discussion and Future Work

While the focus of this paper has been on motivating Nero and demonstrating its practical advantages, this section will discuss some of the more theoretical issues concerning convergence and generalisability of the trained network, as well as possible directions for future work.

6.1. Convergence

Convergence analyses of first order optimisation algorithms typically rely on a model of *smoothness* of the loss function. One of the most commonly encountered models is *Lipschitz smoothness* (Nesterov, 2004), although Sun (2019) points out that this model is somewhat ill-suited to neural networks.

Nero was motivated in Section 3 based on a notion of *layerwise relative smoothness* called *deep relative trust* (Definition 4). Deep relative trust attempts to directly model the smoothness of neural network loss functions, and may be used to derive formal convergence guarantees—see for instance (Bernstein et al., 2020a, Lemma 2).

Yet the empirical success of Nero—which rotates *neurons* through fixed angles—might suggest that there is something missing in a notion of *layerwise* relative smoothness. Indeed the success of Nero might suggest that neural networks are better characterised by a notion of *per-neuron angular smoothness*. For instance, one might surmise that solutions returned by Nero satisfy a notion of angular robustness, as expressed in the following definition:

Definition 6. A solution that attains zero training error is α-*robust* if all neurons may be simultaneously and arbitrarily rotated by up to angle α without inducing an error.

In other words, Definition 6 suggests measuring the sharpness/flatness of converged solutions in terms of an angular parameter α. Such a notion plays a role in generalisation theory, as will be seen in the next section.

6.2. Generalisation

The results in this paper may have a bearing on the generalisation theory of neural systems—an area of research that is still not settled. Consider the following hypothesis:

Hypothesis 1. *Deep learning generalises because SGD is biased towards solutions with small norm.*

This hypothesis is well-known, and is alluded to or mentioned explicitly in many papers (Wilson et al., 2017; Zhang et al., 2017; Bansal et al., 2018; Advani et al., 2020).

But in light of the results in Table 1, Hypothesis 1 encounters some basic problems. First, for some tasks—such as the GAN and translation experiment—SGD simply performs very poorly. And second, Nero is able to find generalising solutions even when the norm of the network is constrained.

For instance, the VGG-11 network and the Wikitext-2 transformer model have no gain parameters so, under Nero, the norm of the weights (though not the biases) is fixed and cannot be "adapting to the data complexity".

Then it seems right to consider an alternative theory:

Hypothesis 2. *Deep learning generalises because the space of networks that fit the training data has large measure.*

This hypothesis is essentially the PAC-Bayesian generalisation theory (McAllester, 1998; Langford & Seeger, 2001) applied to deep learning. Valle-Perez et al. (2019) have developed this line of work, proving the following result:

Theorem 1 (Realisable PAC-Bayes). *First, fix a probability measure $\mathbb{P}$ over the weight space Ω of a classifier. Let S denote a training set of n iid datapoints and let $V_S \subset \Omega$ denote the version space—that is, the subset of classifiers that fit the training data. Consider the population error rate $0 \leqslant \varepsilon(w) \leqslant 1$ of weight setting $w \in \Omega$, and its average over the version space $\varepsilon(V_S) := \mathbb{E}_{w \sim \mathbb{P}}[\varepsilon(w)|w \in V_S]$. Then, for a proportion $1 - \delta$ of random draws of the training set S,*

$$\varepsilon(V_S) \leqslant \ln \frac{1}{1 - \varepsilon(V_S)} \leqslant \frac{\ln \frac{1}{\mathbb{P}[V_S]} + \ln \frac{2n}{\delta}}{n - 1}. \quad (4)$$

The intuition is that for a larger measure of solutions $\mathbb{P}[V_S]$, less information needs to be extracted from the training data to find just *one* solution, thus memorisation is less likely.

A simple bound on $\mathbb{P}[V_S]$ is possible based on this paper's connection between optimisation and architecture, since the problem is reduced to hyperspherical geometry. Consider a balanced network (Definition 2) composed of m neurons each with fan-in d. Then the optimisation domain is isomorphic to the Cartesian product of m hyperspheres:

$$\Omega \cong \underbrace{\mathbb{S}^{d-2} \times ... \times \mathbb{S}^{d-2}}_{m \text{ times}},$$

while $\mathbb{P}$ can be fixed to the uniform distribution on Ω.

Next, suppose that the version space consists of K non-intersecting α-robust solutions (Definition 6). Geometrically, an α-robust solution is the product of m hyperspherical caps. Thus the measure of the version space satisfies:

$$\mathbb{P}[V_S] = K \cdot \mathbb{P}[\text{cap}_{d-2}(\alpha)]^m \geqslant \frac{K}{2^m} \sin^{m(d-2)} \frac{\alpha}{2}, \quad (5)$$

where $\text{cap}_{d-2}(\alpha)$ denotes an α-cap of $\mathbb{S}^{d-2}$, and the inequality follows from (Ball, 1997, Lemma 2.3). Combining Inequality 5 with Inequality 4 yields the following generalisation bound for neural networks:

$$\varepsilon(V_S) \leqslant \frac{m \ln 2 + m(d-2) \ln \frac{1}{\sin \frac{\alpha}{2}} + \ln \frac{2n}{\delta} - \ln K}{n - 1}.$$

Focusing on the dominant terms, the bound suggests that the average test error $\varepsilon(V_S)$ over the space of solutions V_S

is low when the number of datapoints n exceeds the number of parameters md less the entropy $\ln K$ of the multitude of distinct solutions. The theory has two main implications:

1. In the "over-parameterised" regime $md \gg n$, generalisation can still occur if the number of distinct solutions K is exponential in the number of parameters md. In practice, $\ln K$ might be increased relative to md by constraining the architecture based on the symmetries of the data—e.g. using convolutions for image data.

2. All else equal, solutions with larger α-robustness may generalise better. In practice, α might be increased by regularising the training procedure (Foret et al., 2021).

Future work might investigate these ideas more thoroughly.

6.3. Finer-Grained Architectural Awareness

The experiments in this paper suggest that Nero performs well across a wide variety of neural architectures with heterogeneous building blocks—including convolution, attention, embeddings and normalisation layers with gains and biases. Yet the theoretical insights were derived primarily by considering simple neuronal building blocks. A more fine-grained study of different neural network components might yield both improved theoretical understanding and better empirical performance. Indeed this might lead to a more faithful realisation of *neural architecture aware optimisation*.

On a related note, it seems there is scope for new techniques that perform *neural architecture aware regularisation*. The results in Section 5.6 demonstrated the empirical effectiveness of one such technique: regularising gain parameters towards one. Regularising gains towards one is arguably more interpretable than applying weight decay to a flattened vector of all the network weights, and the authors of this paper found it significantly easier to tune.

More generally, in any setting where a deep learning technique operates on a flattened vector of all the network weights, it seems there is a good chance that the technique may be improved by accounting for the neural architecture.

7. Conclusion

This paper has proposed the Nero optimiser based on a combined study of optimisation and neural architecture. Nero pairs two ingredients: (1) projected gradient descent over the space of balanced networks; and (2) per-neuron relative updates. Taken together, a Nero update *turns* each neuron through an angle set by the learning rate.

Nero was found to have strong *out-of-the-box* performance. In almost all the experiments in this paper—spanning GAN training, image classification, natural language processing and reinforcement learning—Nero trained well using its default hyperparameter settings. The two exceptions were the 100 layer MLP and the WMT16 En–De transformer, for which Nero required a reduced learning rate of $\eta = 0.001$. Thus Nero has the potential to accelerate deep learning research and development, since the need for time and energy intensive hyperparameter search may be reduced.

References

Advani, M. S., Saxe, A. M., and Sompolinsky, H. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 2020.

Bai, Y., Wang, Y.-X., and Liberty, E. Proxquant: Quantized neural networks via proximal operators. In *International Conference on Learning Representations*, 2019.

Ball, K. An elementary introduction to modern convex geometry. In *MSRI Publications*, 1997.

Bansal, Y., Advani, M., Cox, D., and Saxe, A. M. Minnorm training: an algorithm for training over-parameterized deep neural networks. *arXiv:1806.00730*, 2018.

Bernstein, J., Vahdat, A., Yue, Y., and Liu, M.-Y. On the distance between two neural networks and the stability of learning. In *Neural Information Processing Systems*, 2020a.

Bernstein, J., Zhao, J., Meister, M., Liu, M.-Y., Anandkumar, A., and Yue, Y. Learning compositional functions via multiplicative weight updates. In *Neural Information Processing Systems*, 2020b.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Review*, 2018.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Brock, A., De, S., and Smith, S. L. Characterizing signal propagation to close the performance gap in unnormalized ResNets. In *International Conference on Learning Representations*, 2021.

Chen, H., Raskutti, G., and Yuan, M. Non-convex projected gradient descent for generalized low-rank tensor regression. *Journal of Machine Learning Research*, 2019.

Chistiakova, M., Bannon, N., Chen, J.-Y., Bazhenov, M., and Volgushev, M. Homeostatic role of heterosynaptic plasticity: models and experiments. *Frontiers in Computational Neuroscience*, 2015.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Neural Information Processing Systems*, 2014.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Neural Information Processing Systems*, 2017.

Huang, L., Liu, X., Liu, Y., Lang, B., and Tao, D. Centered weight normalization in accelerating training of deep neural networks. In *International Conference on Computer Vision*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

Langford, J. and Seeger, M. Bounds for averaging classifiers. Technical report, Carnegie Mellon University, 2001.

Liu, W., Zhang, Y.-M., Li, X., Yu, Z., Dai, B., Zhao, T., and Song, L. Deep hyperspherical learning. In *Neural Information Processing Systems*, 2017.

Liu, W., Liu, Z., Yu, Z., Dai, B., Lin, R., Wang, Y., Rehg, J. M., and Song, L. Decoupled networks. In *Computer Vision and Pattern Recognition*, 2018.

Liu, W., Lin, R., Liu, Z., Rehg, J. M., Paull, L., Xiong, L., Song, L., and Weller, A. Orthogonal over-parameterized training. In *Computer Vision and Pattern Recognition*, 2021.

McAllester, D. A. Some PAC-Bayesian theorems. In *Conference on Computational Learning Theory*, 1998.

Miller, K. and MacKay, D. The role of constraints in Hebbian learning. *Neural Computation*, 1994.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Nesterov, Y. Introductory lectures on convex optimization: A basic course. In *Applied Optimization*, 2004.

Qiao, S., Wang, H., Liu, C., Shen, W., and Yuille, A. Micro-batch training with batch-channel normalization and weight standardization. *arXiv:1903.10520*, 2019.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of Adam and beyond. In *International Conference on Learning Representations*, 2018.

Rochester, N., Holland, J., Haibt, L., and Duda, W. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *Information Theory*, 1956.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Neural Information Processing Systems*, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

Shen, Y., Wang, J., and Navlakha, S. A correspondence between normalization strategies in artificial and biological neural networks. In *From Neuroscience to Artificially Intelligent Systems*, 2020.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Sun, R. Optimization for deep learning: theory and algorithms. *arXiv:1912.08957*, 2019.

Tieleman, T. and Hinton, G. E. Lecture 6.5—RMSprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Turrigiano, G. The self-tuning neuron: Synaptic scaling of excitatory synapses. *Cell*, 2008.

Valle-Perez, G., Camargo, C. Q., and Louis, A. A. Deep learning generalizes because the parameter–function map is biased towards simple functions. In *International Conference on Learning Representations*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Neural Information Processing Systems*, 2017.

von der Malsburg, C. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 1973.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning. In *Neural Information Processing Systems*, 2017.

You, Y., Gitman, I., and Ginsburg, B. Scaling SGD batch size to 32K for ImageNet training. Technical Report UCB/EECS-2017-156, University of California, Berkeley, 2017.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.