Second-Order Complexity Theory for Neural Networks

Yoshihiro Maruyama
Department of Mathematical and Information Sciences
Kyoto University
maruyama@i.h.kyoto-u.ac.jp

Abstract

Machine learning relies on gradient-based training procedures whose empirical efficiency is usually analyzed through finite-dimensional arithmetic counts rather than formal complexity over continuous real-valued data. This paper studies neural-network training primitives as represented-real operators in the second-order complexity framework of Kawamura and Cook. Here, second-order polynomial time (FP_2) means computing a real-valued operator to accuracy 2^{-n} in time polynomial in the requested precision, architecture size, primitive evaluation costs, and analytic bounds encoded with the input. We prove that dense backpropagation, convolutional backpropagation, soft attention, a fixed number of SGD- or Adam-style updates, and related smooth local primitives are FP_2-computable over the relevant represented real spaces. More precisely, dense one-sample backpropagation has tight arithmetic-and-output complexity $\Theta(s)$ for s weights and polynomial bit complexity; convolutional backpropagation is polynomial in the number of active convolution incidences; dense soft-attention backpropagation for sequence length ℓ and model dimension d has arithmetic cost $\Theta(\ell d^2 + \ell^2 d)$ using classical matrix multiplication; and fixed finite optimizer updates remain FP_2-computable, with polynomial precision moduli and denominator bounds determining the required precision. Higher complexity enters through three distinct mechanisms: discontinuous selection, global optimization, and continuous-time flow solving. For example, hard routing, top-k sparsification, argmax decisions, ReLU kink conventions, exact line search, and idealized gradient flow change the second-order complexity of the training operator. Overall, this work provides a rigorous mathematical language for establishing tractability guarantees and identifying computational barriers in neural-network training, thus enabling a unified account of computational complexity for dense networks, CNNs, attention, Transformers, and standard optimizers.

1 Introduction

Gradient-based training is the central computational mechanism underlying machine learning. Its success is commonly analyzed through finite-dimensional arithmetic counts: a forward pass applies the layers, a reverse pass propagates sensitivities, and an optimizer updates the parameters (Goodfellow et al., 2016; Bottou et al., 2018). This perspective has been important for understanding why convolutional networks, residual networks, attention, and Transformer-style models can be trained efficiently in large-scale settings (LeCun et al., 1998; He et al., 2016; Vaswani et al., 2017). However, arithmetic counts alone do not specify how continuous real-valued parameters, data, activations, and losses are computed to a requested accuracy. They characterize the arithmetic structure of the computation, but they often leave precision, conditioning, and branch decisions implicit.

This omission is not merely a matter of numerical accounting. Many training operations that appear similar at the level of a floating-point program differ in their mathematical properties as real-valued

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

operators. Smooth backpropagation, soft attention, and fixed finite optimizer updates are local transformations whose outputs vary continuously with their inputs. By contrast, hard routing, top-k selection, argmax decisions, exact line search, and idealized gradient flow introduce selection, global minimization, or continuous-time evolution. These distinctions matter for theory because a formal complexity statement should separate the part of training that is efficient because it is a local sensitivity computation from the operators whose complexity arises from exact choices, global optima, or solutions of differential equations.

This paper studies neural-network training primitives as operators over represented real spaces, using the second-order complexity framework of Kawamura and Cook (2012). In this setting, the requested output precision is part of the input, and running time may depend polynomially on the precision request, the explicit architecture size, the costs of primitive real functions, and analytic bounds on ranges, derivatives, margins, and denominators encoded with the architecture. The viewpoint builds on computable analysis and type-two complexity (Ko and Friedman, 1982; Ko, 1991; Pour-El and Richards, 1989; Weihrauch, 1993, 2000; Kapron and Cook, 1996; Kawamura and Cook, 2012; Neumann and Steinberg, 2020); for machine learning, its role is to identify which standard training kernels remain efficient when continuous data and precision requirements are treated explicitly.

Our first set of results establishes second-order polynomial-time computability (FP_2) for smooth local training primitives. Dense one-sample backpropagation has tight arithmetic-and-output complexity $\Theta(s)$ for s weights and polynomial bit complexity, extending classical reverse-mode differentiation bounds and modern accounts of automatic differentiation to this framework (Baur and Strassen, 1983; Griewank and Walther, 2008; Baydin et al., 2018). Convolutional backpropagation is also polynomial-time computable, with cost controlled by active convolution incidences (LeCun et al., 1998). Dense soft-attention backpropagation has polynomial bit complexity and arithmetic cost $\Theta(\ell d^2 + \ell^2 d)$ for sequence length ℓ and model dimension d, using classical matrix multiplication (Bahdanau et al., 2015; Vaswani et al., 2017). These results extend to any fixed number of optimizer updates built from these kernels, including SGD- and Adam-style updates (Robbins and Monro, 1951; Kingma and Ba, 2015; Bottou et al., 2018).

These results formalize the standard intuition that local differentiation through a smooth finite network is not the source of higher complexity in this model. The architecture determines the arithmetic work, and the analytic bounds determine the guard bits needed for a certified gradient approximation. Thus the structural mechanism that makes backpropagation practical at fixed precision also supports a represented-real polynomial-time statement.

We then identify three mechanisms that change the complexity of training operators: discontinuous selection, global optimization, and continuous-time flow solving. Hard routing, top-k sparsification, argmax decisions, and ReLU derivative conventions illustrate the role of discontinuous branch choices (Clarke, 1990; Nair and Hinton, 2010; Glorot et al., 2011). Exact line search requires global continuous minimization (Nocedal and Wright, 2006; Boyd and Vandenberghe, 2004), and idealized gradient flow requires solving continuous-time dynamics (Ko, 1983; Kawamura, 2010). For a fixed class of two-dimensional C^2 potentials supplied through potential-value and gradient-evaluation oracles, we prove a second-order polynomial-space upper bound and an exponential query lower bound for gradient-flow endpoint evaluation, establishing $\mathrm{FPSPACE}_2 \setminus \mathrm{FP}_2$ membership.

In this way, second-order complexity theory for continuous structures and operators gives a unified complexity account for dense networks, convolutional networks, attention and Transformer-style blocks, and a fixed number of updates of standard optimizers. It also identifies the operations that move beyond second-order polynomial-time complexity and provides a rigorous mathematical language for the precision and certificate dependencies that floating-point analyses often omit, while preserving the arithmetic-complexity insights that explain much of modern deep learning practice. These quantitative certificates provide practical guidance for choosing working precision, maintaining branch margins, and safeguarding finite optimizer updates. Table 1 in Section 7 summarizes these complexity classifications. Our related work studies the second-order complexity of neural ODE inference (Maruyama, 2026a) and of risk, explanation, and calibration in machine learning (Maruyama, 2026b).

The rest of the paper is organized as follows. Section 2 sets up the second-order complexity model over represented real spaces. Sections 3-6 analyze dense networks, convolution, and attention. Section 7 classifies optimizer updates and harder operators. The appendices contain proofs, additional results, and concrete complexity illustrations, as well as additional related work.

2 Complexity over continuous structures

We use complexity theory over represented spaces to analyze algorithms whose inputs and outputs are real-valued objects. This is the natural setting for neural-network optimization: a training step is not merely a finite floating-point program, but an operator that maps real parameters, real data, and a requested precision to an approximation of another real object. The framework of Kawamura and Cook formalizes this viewpoint by representing continuous objects through oracle names and bounding oracle-machine running time (Kawamura and Cook, 2012). We recall the basic ingredients of the Kawamura-Cook framework needed for neural-network operators.

An oracle name provides access to a real input through finite approximations at any requested accuracy. A precision request n asks for absolute error at most 2^{-n}, so n measures accuracy in bits. The term "second-order" refers to running-time bounds that take the oracle's answer-length function as an argument, together with the finite input size and requested precision.

Let $\Sigma = \{0, 1\}$ and let Σ^* be the set of finite binary strings. Following the standard second-order presentation, names are string functions. For a string function $\varphi : \Sigma^* \to \Sigma^*$, write $|\varphi|(n) := \max\{|\varphi(a)| : |a| \le n\}$. Let $\mathbb{D} := \{m2^{-k} : m \in \mathbb{Z},\ k \in \mathbb{N}\}$ be the dyadic rationals, encoded by signed binary mantissas and binary exponents. A second-order polynomial is an expression $P(\lambda, n)$ built from constants, n, addition, multiplication, and applications of the unary function argument λ to previously built expressions; in runtime bounds below $\lambda = |\varphi|$ is the length function of an oracle name.

A represented space is a pair (X, δ_X) where $\delta_X :\subseteq (\Sigma^*)^{\Sigma^*} \to X$ is a partial surjection. For real numbers, a name answers a unary precision query 1^n by a dyadic q_n satisfying $|q_n - x| \le 2^{-n}$. Finite numerical inputs are named coordinatewise. Their shape and the padded finite parameter code κ described below are supplied as discrete input. A full-output precision query requests all coordinates with maximum error at most 2^{-n}; the padded architecture code has length at least the number of coordinates to be materialized.

Definition 1 (Second-order polynomial-time computability)**.** Let (X, δ_X) and (Y, δ_Y) be represented spaces. A partial map $F :\subseteq X \to Y$ is (δ_X, δ_Y)-FP_2-computable if an oracle machine realizes F within a second-order polynomial time bound. Here, realizing F means converting every δ_X-name of $x \in \mathrm{dom}(F)$ into a δ_Y-name of $F(x)$.

The machine is charged for writing precision queries and for reading and processing oracle answers. Fixed finite products and fixed finite compositions preserve FP_2-computability (Kawamura and Cook, 2012). An input-dependent number of compositions additionally requires uniform bounds on the number of steps, intermediate magnitudes, and precision propagation.

The architecture determines the arithmetic operation count, while range and derivative bounds control operand lengths and error amplification at the requested accuracy. The input bundle κ records the architecture and these analytic bounds, which we call certificates. Its architecture component contains the exact descriptors and unary padding $1^{S_{\mathrm{arch}}}$, where S_{arch} bounds the explicit coordinate and incidence counts used by the algorithm. All requested input and output tensor sizes are bounded by a polynomial in S_{arch}.

Numerical certificates remain binary-encoded dyadics. In addition, every nonnegative upper bound B is accompanied by a unary scale 1^{r_B} with $r_B \ge \lceil \log_2(2 + B) \rceil$, and every positive lower bound γ by 1^{r_γ} with $r_\gamma \ge \lceil \log_2(2 + 1/\gamma) \rceil$. The parameter length includes the entire dyadic codes, architecture descriptors, and padding. Validity of the analytic certificates defines the promised input domain. Thus polynomial dependence on the supplied scales is polynomial dependence on actual finite input length. Upper bounds are rounded upward and positive lower bounds downward to dyadic powers of two using their unary scales: $B \le 2^{r_B}$ and $2^{-r_\gamma} \le \gamma$. Derived product and precision bounds are evaluated by arithmetic on the resulting integer scales.

Definition 2 (Parameterized represented neural operators)**.** A family $\{F_\kappa\}_\kappa$ is uniformly FP_2-computable if one oracle machine, on every valid padded code κ, numerical-input name φ of $x \in \mathrm{dom}(F_\kappa)$, and precision request 1^n, outputs all coordinates of $F_\kappa(x)$ to maximum coordinate error 2^{-n} within time $P(|\varphi|, n + |\kappa|)$ for one second-order polynomial P.

Equivalently, the finite code κ can be incorporated into a combined input name. The full-output and coordinate-query formulations are polynomial-time equivalent under the explicit-size padding convention above.

A continuity (or precision) modulus for a map f is a function $\mu : \mathbb{N} \to \mathbb{N}$ such that $\|u-v\|_\infty \le 2^{-\mu(n)}$ implies $\|f(u) - f(v)\|_\infty \le 2^{-n}$ on its certified domain; polynomial precision moduli are bounded by one polynomial in $n + |\kappa|$ uniformly over the family. An evaluation name for a continuous scalar- or vector-valued function f on a dyadic box consists of a continuity modulus and an oracle that, on query $(a, 1^n)$, returns a dyadic approximation to $f(a)$ with maximum error 2^{-n}, where a is a dyadic point in the box.

For a straight-line program, a program certificate specifies a dyadic box C_j for the argument tuple of each gate g_j, containing the exact gate arguments on the promised input domain, and a Lipschitz bound L_j for g_j on C_j in the maximum norm. A magnitude bound $B_{\mathrm{prog}} \ge 1$ covers numerical inputs, box endpoints, constants, and all gate values on the certified boxes. These bounds may be supplied or derived uniformly from the layer envelopes. Their scales use the padding convention above. Approximate argument tuples are clipped coordinatewise to C_j before evaluation. Clipping leaves the exact computation unchanged and is nonexpansive. Signed reciprocals use a certified positive or negative interval separated from zero. Lemma 4 proves the resulting error bound. Primitives such as square root at zero are handled separately through their explicit precision moduli.

Let p denote fractional working precision and let $M(w)$ denote the bit cost of an elementary arithmetic operation on operands of total bit length at most w. Let $c_\kappa \le |\kappa|$ bound the code lengths of numerical certificate endpoints and constants. For stored magnitudes bounded by $B_{\mathrm{prog}} + 1$, use

$$w = \max\{p + \lceil \log_2(2 + B_{\mathrm{prog}}) \rceil, c_\kappa\} + O(1).$$

One may take $M(w) = O(w^2)$ in the classical arithmetic model. We report arithmetic operations and primitive calls separately, combining their evaluation costs with oracle access and input/output in the bit-runtime bounds. Primitive costs $T_g(p; \kappa, w)$ include evaluation to absolute error 2^{-p} on the certified domain, with arguments of length $O(w)$, and must be uniform in the encoded primitive and certificate data. We write T_{io} for finite-code decoding, numerical-oracle query and answer processing, and output writing. Polynomial-size programs give uniform FP_2 bounds when their primitive costs, intermediate word lengths, and required precision are polynomially bounded in the actual input parameters.

Lemma 1 (Uniform closure for certified neural programs)**.** *Let a family of straight-line programs have a polynomial number of gates in the explicit architecture size. Suppose its gate evaluators are uniformly polynomial time, its certified intermediate magnitudes have polynomially bounded logarithms, and $\sum_j \log_2(1 + L_j)$ is polynomially bounded in the padded parameter length. Then the represented operator computed by the program is uniformly FP_2-computable. A fixed finite number of additional polynomial-time primitives with polynomial precision moduli also preserves FP_2-computability.*

3 Dense neural-network model

We now instantiate the framework for dense networks. The architecture is specified by exact finite data, the numerical inputs are Cauchy-named real coordinates, and κ records the bounds needed to run the finite computation at a chosen precision.

Fix a depth L and widths $d_0, \dots, d_L$. A dense feedforward network has parameters

$$\theta = \{(W^\ell, b^\ell)\}_{\ell=1}^{L}, \qquad W^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \quad b^\ell \in \mathbb{R}^{d_\ell}. \tag{1}$$

Given an input $x \in \mathbb{R}^{d_0}$, define

$$a^0 = x, \qquad z^\ell = W^\ell a^{\ell-1} + b^\ell, \qquad a^\ell = \sigma_\ell(z^\ell), \qquad 1 \le \ell \le L, \tag{2}$$

where each σ_ℓ acts coordinatewise. For a label or target y, the scalar objective is

$$J(\theta; x, y) := \mathcal{L}(a^L, y). \tag{3}$$

We write

$$s := \sum_{\ell=1}^{L} d_\ell d_{\ell-1}, \qquad N := s + \sum_{\ell=1}^{L} d_\ell, \tag{4}$$

for the number of weights and total trainable scalars, respectively.

For a finite tensor, $\|\cdot\|_\infty$ denotes the maximum absolute entry. For a matrix or linear map A, write $\|A\|_{p\to q} := \sup_{\|v\|_p\le 1} \|Av\|_q$ for the induced operator norm. The following norm bounds control forward and backward propagation:

$$r_\ell := \left\|W^\ell\right\|_{\infty\to\infty} = \max_i \sum_j |W^\ell_{ij}|, \qquad c_\ell := \left\|W^\ell\right\|_{1\to 1} = \max_j \sum_i |W^\ell_{ij}|. \tag{5}$$

Let A_0 be an upper bound for $\|x\|_\infty$. Recursively define forward range bounds

$$Z_\ell := r_\ell A_{\ell-1} + \left\|b^\ell\right\|_\infty, \qquad A_\ell := \sup_{|t|\le Z_\ell} |\sigma_\ell(t)|. \tag{6}$$

For smooth activations, let

$$S_\ell := \sup_{|t|\le Z_\ell} |\sigma'_\ell(t)|, \qquad T_\ell := \sup_{|t|\le Z_\ell} |\sigma''_\ell(t)|. \tag{7}$$

Finally, on the bounded output-label domain, write $w = (u, y)$, where u ranges over the output space of a^L. When labels are represented real coordinates, let Y_0 be a dyadic upper bound for $\|y\|_\infty$ on the label domain. Assume loss envelopes

$$U := \sup_{(u,y)} \|\nabla_u \mathcal{L}(u, y)\|_1, \qquad V := \sup_w \left\|\nabla^2_{ww}\mathcal{L}(w)\right\|_{\infty\to 1}. \tag{8}$$

If labels are discrete rather than represented real coordinates, w is understood to omit the label coordinates from the Hessian envelope. For represented real labels, this Hessian envelope also controls the dependence of $\nabla_\theta J$ on y. The displayed quantities are exact norms or suprema. In κ, the same symbols denote dyadic upper bounds on these quantities. These encoded bounds enter the working-precision and runtime estimates.

Definition 3 (The represented class $\mathsf{NN}^{C^2}_\kappa$)**.** The class $\mathsf{NN}^{C^2}_\kappa$ consists of finite dense feedforward networks whose real parameters and data are given by Cauchy names. Primitive codes identify σ_ℓ, σ'_ℓ, $\mathcal{L}$, and $\nabla_u\mathcal{L}$ from a fixed catalogue with uniform polynomial-time evaluation on the certified boxes. Bounds on second derivatives control precision propagation in the backpropagation analysis. The padded bundle records $L, (d_\ell)_{\ell=0}^L$, the label-domain code, A_0, Y_0 for represented real labels, $(r_\ell, c_\ell, \|b^\ell\|_\infty)$, $(A_\ell, Z_\ell, S_\ell, T_\ell)$, and U, V as valid dyadic upper bounds. Bounds used for derivative-based input moduli hold on a common convex numerical-input domain. The forward and reverse programs also use the program certificates of Definition 2, including ranges for partial sums, primitive values, adjoints, and gradient outputs. Lipschitz bounds for activation and activation-derivative gates are supplied by S_ℓ and T_ℓ, and those for loss-gradient gates by V. For forward evaluation of the scalar loss, the catalogue supplies its uniform polynomial precision modulus, or a certified gate Lipschitz bound. Discrete labels are encoded exactly; all represented label coordinates are included in the numerical-input representation.

For dense networks, this separates the two quantities that later appear in the runtime: the scalar-operation count is determined by the architecture, while the working precision is determined by the analytic scale of the envelopes. This matches numerical practice: a backpropagation kernel has one cost at fixed precision, while its exact-real bit complexity also depends on conditioning.

By Lemma 1, the certified dense forward map $(\theta, x, y) \mapsto J(\theta; x, y)$ and certified compositions of layers, residual additions, and fixed-shape linear maps are uniformly FP_2-computable.

For the bit-complexity analysis, let H_{dense} denote the explicit bound derived in Appendix A.3 on every scalar second partial derivative of J with respect to the numerical inputs, including parameters, input data, and represented label coordinates, on the certified common convex domain. This bound controls the gradient's sensitivity to its original inputs, while the program certificates control amplification of errors introduced during forward and reverse computation.

4 Backpropagation is second-order polynomial-time

We now prove the dense-network classification. Backpropagation (Werbos, 1974; Rumelhart et al., 1986) is the reverse-mode differentiation algorithm for the scalar objective $J(\theta; x, y)$, and its constant-factor relation to forward evaluation is consistent with the classical arithmetic-circuit bounds for

partial derivatives (Wengert, 1964; Linnainmaa, 1976; Baur and Strassen, 1983; Speelpenning, 1980; Rall, 1981) and the modern automatic-differentiation view of backpropagation (Baydin et al., 2018). The objective is to state the result as an operator over represented reals, with explicit precision dependence.

Let $\delta^\ell := \partial J/\partial z^\ell \in \mathbb{R}^{d_\ell}$ denote the preactivation adjoint at layer ℓ, and let $\odot$ denote coordinatewise product. The backpropagation recurrence is

$$\delta^L = \nabla_u \mathcal{L}(a^L, y) \odot \sigma_L'(z^L), \tag{9}$$

$$\delta^\ell = \big((W^{\ell+1})^\top \delta^{\ell+1}\big) \odot \sigma_\ell'(z^\ell), \qquad 1 \le \ell < L, \tag{10}$$

with parameter gradients

$$\nabla_{W^\ell} J = \delta^\ell (a^{\ell-1})^\top, \qquad \nabla_{b^\ell} J = \delta^\ell. \tag{11}$$

Theorem 1 (Arithmetic complexity of dense backpropagation)**.** *For a dense feedforward network with $s = \sum_{\ell=1}^{L} d_\ell d_{\ell-1}$ weights, one-sample backpropagation computes the full parameter gradient using*

$$C_{\mathrm{bp}}^{\mathrm{arith}} = 2s + 2\sum_{\ell=1}^{L-1} d_\ell d_{\ell+1} + s + O\left(\sum_{\ell=1}^{L} d_\ell\right) \tag{12}$$

scalar arithmetic operations, plus primitive calls for the activations, activation derivatives, and loss gradient. In particular,

$$C_{\mathrm{bp}}^{\mathrm{arith}} \le 5s + O\left(\sum_{\ell=1}^{L} d_\ell\right), \tag{13}$$

so the displayed implementation uses $O(s)$ scalar arithmetic operations, in addition to these primitive calls. Counting one unit for each scalar output write gives a tight $\Theta(s)$ arithmetic-and-output bound for materializing the full parameter gradient.

This bound shows that the reverse pass has scalar arithmetic cost within a constant factor of forward evaluation. The following theorem establishes second-order polynomial-time computability by choosing working precision that controls both input approximation and roundoff accumulation.

Theorem 2 (Bit complexity and second-order polynomial time)**.** *Let $\mathcal{N} \in \mathsf{NN}_\kappa^{C^2}$ and let*

$$\mathrm{BP}(\theta, x, y) := \nabla_\theta J(\theta; x, y). \tag{14}$$

Let d_y be the number of represented real label coordinates, with $d_y = 0$ for discrete labels, and set

$$q := N + d_0 + d_y. \tag{15}$$

For the certified forward/reverse program, let A_{bp} be its gate count, A_{arith} its arithmetic-gate count, and define

$$K_{\mathrm{bp}} := \prod_{j=1}^{A_{\mathrm{bp}}} (1 + L_j), \qquad C_{\mathrm{bp}} := 1 + qH_{\mathrm{dense}} + (A_{\mathrm{bp}} + 1)K_{\mathrm{bp}}.$$

A sufficient fractional precision and stored word length are

$$\begin{aligned} p &= n + \lceil \log_2(2C_{\mathrm{bp}}) \rceil, \\ w &= \max\{p + \lceil \log_2(2 + B_{\mathrm{bp}}) \rceil, c_\kappa\} + O(1). \end{aligned} \tag{16}$$

where B_{bp} is the program magnitude bound. The gradient is uniformly FP_2-computable, with

$$\begin{aligned} T_{\mathrm{bp}}(n) = O\Big(& T_{\mathrm{io}} + A_{\mathrm{arith}} M(w) + T_{\nabla_u \mathcal{L}}(p; \kappa, w) \\ &+ \sum_{\ell=1}^{L} d_\ell \big(T_{\sigma_\ell}(p; \kappa, w) + T_{\sigma_\ell'}(p; \kappa, w)\big)\Big). \end{aligned} \tag{17}$$

Here $A_{\mathrm{arith}} \le 5s + O(\sum_\ell d_\ell)$ for the arithmetic trace; clipping and rounding contribute a constant-factor execution overhead. The gate count, $\log K_{\mathrm{bp}}$, $\log(2 + B_{\mathrm{bp}})$, and primitive costs are polynomially controlled by the explicit architecture and padded certificates. The term qH_{dense} describes input sensitivity; the program term controls finite-precision execution, including evaluations at approximate arguments.

Appendix B.3 provides numerical architecture examples and discusses the precision and arithmetic-cost bounds.

Remark 1 (Why smoothness and parameters matter)**.** Smooth activations and losses give a continuous gradient operator with an explicit modulus controlled by H_{dense}. If a nonsmooth primitive such as ReLU, max-pooling, argmax, or top-k routing is used with a single-valued derivative convention, the exact-real gradient map is generally discontinuous at kink or tie surfaces and therefore cannot be globally computable. On strata specified by a positive margin, branch identification followed by evaluation of the corresponding smooth program gives piecewise FP_2-computability.

5 Convolutional networks

Convolutional networks can be treated in the same framework as follows. A layer applies a sparse linear operator with shared coefficients and adds a bias. The kernel and bias are given by Cauchy names.

Definition 4 (Convolutional parameter bundle)**.** Let m be the spatial dimension; H_j, H'_j, k_j the input, output, and kernel sizes along axis j; and $C_{\mathrm{in}}, C_{\mathrm{out}}$ the input and output channel counts. For a convolutional network or layer, κ_{conv} consists of the exact integer geometry data

$$m,\ (H_j)_{j=1}^m,\ (H'_j)_{j=1}^m,\ (k_j)_{j=1}^m,\ C_{\mathrm{in}},\ C_{\mathrm{out}}, \tag{18}$$

the padding, stride, dilation, grouping pattern, and partial input-position map π. The architecture uses explicit coordinate/incidence padding. Analytic certificates include input and bias bounds, sparse-operator and transpose norms, activation and loss envelopes, and a bound H_{conv} on all scalar second partials of the objective on a common convex numerical-input domain. The bundle also supplies or uniformly derives argument boxes, gate Lipschitz bounds, and an intermediate magnitude bound for the entire forward/reverse program, including partial sums and shared-weight gradient accumulation. All scales are encoded as in Definition 2.

Consider one m-dimensional convolutional layer with input

$$X \in \mathbb{R}^{H_1 \times \cdots \times H_m \times C_{\mathrm{in}}}, \qquad W \in \mathbb{R}^{k_1 \times \cdots \times k_m \times C_{\mathrm{in}} \times C_{\mathrm{out}}}, \tag{19}$$

and output spatial size $H'_1 \times \cdots \times H'_m$. Let

$$I := \prod_{j=1}^m H_j, \qquad P := \prod_{j=1}^m H'_j, \qquad K := \prod_{j=1}^m k_j, \tag{20}$$

where I is the number of input positions, P is the number of output positions, and K is the kernel volume. For fixed convolutional geometry, each output preactivation has the form

$$z_{p,o} = b_o + \sum_{r=1}^{K} \sum_{c=1}^{C_{\mathrm{in}}} W_{r,c,o}\, X_{\pi(p,r),c}, \tag{21}$$

where p indexes output position, o indexes output channel, r indexes kernel offsets, c indexes input channels, π is the discrete input-position map, and invalid padded indices contribute zero.

Theorem 3 (Convolutional backpropagation)**.** *For a convolutional layer with input-position count I, output-position count P, kernel volume K, input channels C_{in}, and output channels C_{out}, forward evaluation has arithmetic complexity*

$$C_{\mathrm{fwd}}^{\mathrm{conv}} \le 2PKC_{\mathrm{in}}C_{\mathrm{out}} + O(PC_{\mathrm{out}}), \tag{22}$$

and one-sample backpropagation through the layer has arithmetic complexity

$$C_{\mathrm{bp}}^{\mathrm{conv}} \le 6PKC_{\mathrm{in}}C_{\mathrm{out}} + O(PC_{\mathrm{out}} + IC_{\mathrm{in}} + KC_{\mathrm{in}}C_{\mathrm{out}} + C_{\mathrm{out}})\,. \tag{23}$$

Let q_{conv} count all varying numerical inputs, A_{conv} count the gates of the certified program, $K_{\mathrm{conv}} = \prod_j (1 + L_j)$, and B_{conv} bound its intermediate magnitudes. Then convolutional backpropagation is uniformly FP_2-computable. A sufficient precision is

$$p = n + \lceil \log_2(2\,[1 + q_{\mathrm{conv}}H_{\mathrm{conv}} + (A_{\mathrm{conv}} + 1)K_{\mathrm{conv}}]) \rceil\,. \tag{24}$$

The bit bound uses $w = \max\{p + \lceil \log_2(2 + B_{\mathrm{conv}}) \rceil, c_\kappa\} + O(1)$, the displayed arithmetic count times $M(w)$, primitive-evaluation costs, and oracle/input/output costs. The arithmetic count controls structural work, and the precision bound controls amplification of local program errors.

The range and precision bounds in κ_{conv} also yield uniform FP_2-computability for grouped and depthwise-separable convolutions, with structural costs determined by active local multiply-accumulate incidences and full output materialization; Appendix B.1 gives the explicit bounds.

6 Attention and Transformer blocks

Soft attention likewise fits the framework for compositions of smooth primitives, but its dominant structural cost is global interaction rather than local support. Let ℓ_{q} and ℓ_{kv} be the query and key/value lengths, and let d_k, d_v be the query/key and value feature dimensions. For query, key, and value matrices

$$Q \in \mathbb{R}^{\ell_{\mathrm{q}} \times d_k}, \qquad K \in \mathbb{R}^{\ell_{\mathrm{kv}} \times d_k}, \qquad V \in \mathbb{R}^{\ell_{\mathrm{kv}} \times d_v},$$

scaled dot-product attention (Vaswani et al., 2017) is

$$\mathrm{Attn}(Q, K, V) := \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \tag{25}$$

where softmax is applied rowwise. In multi-head self-attention, for $X \in \mathbb{R}^{\ell \times d}$ and h heads with $d = hd_k = hd_v$, the exact discrete architecture data are ℓ, d, h, d_k, d_v and the projection matrices have shapes $W_a^Q, W_a^K \in \mathbb{R}^{d \times d_k}$, $W_a^V \in \mathbb{R}^{d \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d}$. For head a,

$$\begin{aligned} Q_a &= XW_a^Q, & K_a &= XW_a^K, & V_a &= XW_a^V, \\ \Pi_a &= \mathrm{softmax}\left(\frac{Q_a K_a^\top}{\sqrt{d_k}}\right), & O_a &= \Pi_a V_a. \end{aligned}$$

The head outputs are concatenated and multiplied by the output projection W^O.

Definition 5 (Attention parameter bundle)**.** For a soft-attention block, κ_{attn} contains the architecture ℓ, d, h, d_k, d_v with explicit-size padding and a fixed mask whose softmax rows are nonempty. Dyadic bounds cover the input and projection matrices, score and value ranges, derivatives of the loss with respect to outputs and represented label coordinates, and any layer-normalization radicand margin. A bound H_{attn} covers all scalar second partials of the objective on a common convex numerical-input domain. The bundle also supplies or uniformly derives gate argument boxes, gate Lipschitz constants, and intermediate magnitude bounds for the forward and reverse programs, including adjoints and partial sums. Softmax is evaluated with a rowwise maximum shift so exponential values stay bounded and the normalization denominator is bounded below. These bounds use the padded encoding of Definition 2.

Rowwise softmax has uniformly bounded first and second partial derivatives and is uniformly FP_2-computable on bounded score boxes, with row length included in the explicit architecture size (Lemma 8 in Appendix A.8).

Theorem 4 (Soft-attention backpropagation)**.** *Using classical matrix multiplication, dense multi-head self-attention with sequence length ℓ, model dimension d, and h heads satisfying $d = hd_k = hd_v$ has forward arithmetic cost*

$$C_{\mathrm{fwd}}^{\mathrm{attn}} \le 8\ell d^2 + 4\ell^2 d + O(h\ell^2), \tag{26}$$

and one-sample backpropagation has arithmetic complexity

$$C_{\mathrm{bp}}^{\mathrm{attn}} \le 24\ell d^2 + 12\ell^2 d + O(h\ell^2). \tag{27}$$

The scalar arithmetic cost of this implementation is $\Theta(\ell d^2 + \ell^2 d)$. Under the certified parameter bundle, let q_{attn} count numerical inputs, A_{attn} count program gates, $K_{\mathrm{attn}} = \prod_j (1 + L_j)$, and B_{attn} bound intermediate magnitudes. A sufficient precision is

$$p = n + \lceil \log_2(2\,[1 + q_{\mathrm{attn}} H_{\mathrm{attn}} + (A_{\mathrm{attn}} + 1)K_{\mathrm{attn}}]) \rceil\,. \tag{28}$$

Use $w = \max\{p + \lceil \log_2(2 + B_{\mathrm{attn}}) \rceil, c_\kappa\} + O(1)$ in the arithmetic bit costs and add primitive and oracle/input/output costs. These quantities are polynomially controlled by the padded certificates, proving uniform FP_2-computability.

Appendix B.3 provides numerical architecture examples and discusses the precision and arithmetic-cost bounds.

The common attention variants follow by replacing the dense $\ell \times \ell$ interaction graph. For cross-attention with query length ℓ_{q} and key-value length ℓ_{kv},

$$C_{\mathrm{bp}}^{\mathrm{xattn}} \le 12(\ell_{\mathrm{q}} + \ell_{\mathrm{kv}})d^2 + 12\ell_{\mathrm{q}}\ell_{\mathrm{kv}} d + O(h\ell_{\mathrm{q}}\ell_{\mathrm{kv}}). \tag{29}$$

For windowed or fixed sparse attention in which every query attends to at most r keys,

$$C_{\mathrm{bp}}^{\mathrm{win}} \le 24\ell d^2 + 12r\ell d + O(hr\ell). \tag{30}$$

Causal masks and fixed sparse masks are treated by applying rowwise softmax only over the allowed nonempty index set, rather than inserting $-\infty$ as a real input.

A Transformer block combines multi-head attention, residual additions, layer normalization (Ba et al., 2016), and positionwise feedforward layers. For $x \in \mathbb{R}^d$, set $\mu(x) = d^{-1}\sum_{i=1}^d x_i$ and $\sigma^2(x) = \mathrm{Var}(x) = d^{-1}\sum_{i=1}^d (x_i - \mu(x))^2$. For layer normalization, a positive bound $\sigma^2(x) + \epsilon \ge \gamma > 0$ controls the precision of reciprocal-square-root evaluation; a fixed $\epsilon > 0$ provides such a bound with $\gamma = \epsilon$. Bounds on the learned scale and shift parameters enter the same precision estimate. Fixed compositions preserve FP_2-computability, and variable-depth stacks satisfy the same conclusion when the full program meets Lemma 1.

7 Optimization-method classification

We now turn from differentiation kernels to optimizer updates. These bounds quantify the cost of executing prescribed updates; a convergence analysis additionally determines the number of updates required to reach a specified optimization tolerance. The analysis also covers input-dependent iteration counts when the complete computation satisfies the uniform size, range, and precision bounds detailed below. The following results also characterize exact branch selection, exact minimizer selection, and continuous-time endpoint evaluation.

Theorem 5 (Fixed finite optimizer updates)**.** *For a smooth neural-network family satisfying the parameter-bundle assumptions above, a composition of a fixed number of backpropagation calls and uniformly polynomial-time vector primitives with supplied polynomial precision moduli is* FP_2*-computable.*

For $B \ge 1$, let $\bar{g} = B^{-1}\sum_{i=1}^B \nabla_\theta J(\theta; x_i, y_i)$ and suppose each sample gradient has maximum norm at most G. A sufficient cost bound for computing this mini-batch gradient mean is

$$\begin{aligned} p_B &= n + \lceil \log_2 B \rceil + 2, \\ w_B &= p_B + \lceil \log_2(2 + B(G+1)) \rceil + O(1), \\ T_{\mathrm{mb}}(n) &\le \sum_{i=1}^B T_{\mathrm{bp},i}(p_B) + O(BNM(w_B)), \end{aligned} \tag{31}$$

with input/output costs included in the gradient evaluation bounds. If T_{bp} is a common worst-case bound, the sum is at most $BT_{\mathrm{bp}}(p_B)$.

For the SGD update $\theta^+ = \theta - \eta\bar{g}$, supply $|\eta| \le \eta_{\max}$ and $\|\bar{g}\|_\infty \le G$. Approximating θ, η, and $\bar{g}$ to

$$p_{\mathrm{upd}} = n + \lceil \log_2(8(1 + \eta_{\max} + G)) \rceil$$

and performing the final multiply/subtract with corresponding guard bits suffices. Its cost includes $T_{\mathrm{mb}}(p_{\mathrm{upd}})$, learning-rate and parameter access, and the final $O(N)$ arithmetic operations at the appropriate total word length.

For SGD, the realized sample or mini-batch indices are treated as discrete input.

The theorem covers updates of SGD, momentum, Nesterov momentum, RMSProp, and Adam. At iteration $t \ge 1$, let $\beta_1, \beta_2 \in [0, 1)$ be the decay parameters and let $\widehat{m}$ and $\widehat{v} \ge 0$ be the bias-corrected first and second gradient moments. Standard Adam uses $\widehat{m}/(\sqrt{\widehat{v}} + \epsilon)$ coordinatewise (Kingma and Ba, 2015). The denominator is bounded below by $\epsilon > 0$, while evaluation at zero second moments is supported by the square-root modulus $|\sqrt{a} - \sqrt{b}| \le \sqrt{|a - b|}$ for $a, b \ge 0$, so a square-root output request of r bits uses $2r + O(1)$ input bits. Approximate radicands may be clipped to $[0, \infty)$. When hyperparameters vary, lower bounds for ϵ and the bias-correction denominators $1 - \beta_i^t$ for $i = 1, 2$ form part of κ, together with state and gradient bounds. These bounds and the square-root modulus determine the precision needed to evaluate each update.

Hessian-vector products are likewise uniformly FP_2-computable under certified C^3 bounds on a common convex domain, input and direction range bounds, uniformly polynomial-time evaluators for

the required primitive derivatives, and full-program certificates satisfying Lemma 1. Appendix B.2 develops the forward-over-reverse construction, its sensitivity bound, and its precision and runtime estimates.

The same finite-composition result covers Newton-CG, Gauss-Newton, and quasi-Newton updates with appropriate primitive and nondegeneracy certificates. For an iteration count k supplied in unary, uniform FP_2-computability follows from a full iteration trace whose gate count, accumulated log-amplification, and word length are polynomially bounded in $n + |\kappa| + k$, with valid range and denominator bounds at every iterate and the same uniform primitive-evaluation assumptions. For non-Lipschitz primitives, polynomially bounded composed precision moduli provide the corresponding precision control.

We next consider exact minimizer selection from objective-value access and gradient-flow endpoint evaluation from gradient access, under the representations specified below. We write $\mathrm{FPSPACE}_2$ for oracle computation using space bounded by a second-order polynomial in the name-length function and $n + |\kappa|$, counting the query and answer workspaces. Exact line search below means a single-valued selection of an exact global minimizer on a compact interval from evaluation access to a continuous objective with a supplied continuity modulus.

Oracle gradient-flow evaluation means the endpoint operator $E_T(V, x_0) = X(T)$, where $X' = -\nabla V(X)$ and $X(0) = x_0$. Its input representation supplies the dimension explicitly, a dyadic box containing the promised trajectory, evaluation access to the gradient on that box, initial data, a dyadic time horizon T, magnitude and Lipschitz bounds B, L, and unary 1^Λ with $\Lambda \geq TL$. Numerical magnitude scales use the earlier padded convention. Potential-value access may also be supplied.

Table 1: Second-order complexity classification of training operators.

Operator	Classification	Source of behavior
Dense backpropagation	FP_2	certified forward/reverse program
Convolutional backpropagation	FP_2	certified sparse computation
Soft-attention backpropagation	FP_2	certified normalized computation
Fixed finite optimizer updates	FP_2	certified finite composition
ReLU/max-pool derivative	FP_2 with margins	branch identification
Hard support selection	FP_2 with gaps	support identification from ranking gaps
Exact minimizer selection	noncomputable in general	discontinuity at ties
Oracle gradient-flow endpoint	$\mathrm{FPSPACE}_2 \setminus \mathrm{FP}_2$	continuous-time integration

A sign/ranking margin of $\gamma > 0$ enables branch identification by approximating each decision value to error at most $\gamma/4$. For hard attention and top-k routing, this identifies the active support. The margin, requested output accuracy, and program error bounds together determine the working precision for identifying and evaluating the selected smooth branch. Evaluating the selected branch program then yields the corresponding derivative-selection operator in FP_2. Exact minimizer selection has a discontinuity already on the affine family $f_a(t) = at$ on $[0, 1]$, yielding Proposition 4 in Appendix B.5.

For gradient-flow endpoints, Theorem 6 in Appendix B.6 gives a polynomial-space algorithm with the unary conditioning parameter $\Lambda \geq TL$. Theorem 7 in Appendix B.6 proves an exponential point-query lower bound for a fixed class of two-dimensional C^2 potentials with both potential and gradient access. These results establish $\mathrm{FPSPACE}_2 \setminus \mathrm{FP}_2$ membership for the represented endpoint operator. The classification thus connects efficient local neural computations with precise selection and continuous-time complexity results.

8 Conclusion

We have established second-order polynomial-time bounds for backpropagation, convolution, soft attention, and fixed finite optimizer updates with polynomial precision moduli, explicitly accounting for output accuracy and the analytic bounds in the input bundles. The harder cases arise not from local differentiation, but from discontinuous selection, global optimization, and continuous-time flow solving. The oracle gradient-flow endpoint operator belongs to $\mathrm{FPSPACE}_2 \setminus \mathrm{FP}_2$ already for two-dimensional C^2 potentials with fixed initial conditions, time horizon, and derivative bounds.

Acknowledgments and Disclosure of Funding

This work was supported by JST and JSPS. There are no competing financial or commercial interests; this is purely theoretical work on the computational complexity of neural-network training primitives.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey A. Radul, and Jeffrey M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18:1–43, 2018.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1998.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Frank H. Clarke. *Optimization and Nonsmooth Analysis*. SIAM, 1990.

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.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Second edition, SIAM, 2008.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, 2015.

Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM Journal on Computing*, 25:117–132, 1996.

Akitoshi Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19:305–332, 2010.

Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. *ACM Transactions on Computation Theory*, 4:Article 5, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ker-I Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58:157–194, 1983.

Ker-I Ko. *Complexity Theory of Real Functions*. Birkhäuser, 1991.

Ker-I Ko and Harvey Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20:323–352, 1982.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010.

Yoshihiro Maruyama. Second-order complexity of neural ODE inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026a.

Yoshihiro Maruyama. Second-order complexity theory for risk, explanation, and calibration in machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026b.

Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Sciences*, 12:147–178, 1976.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

Yurii E. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Eike Neumann and Florian Steinberg. Parametrised second-order complexity theory with applications to the study of interval computation. *Theoretical Computer Science*, 806:281–304, 2020.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Second edition, Springer, 2006.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6:147–160, 1994.

Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 1964.

Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*. Springer, 1989.

Louis B. Rall. *Automatic Differentiation: Techniques and Applications*. Springer, 1981.

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

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14:1723–1738, 2002.

B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1980.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude. *Neural Networks for Machine Learning*. Coursera lecture slides, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017.

Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113:191–210, 1993.

Klaus Weihrauch. *Computable Analysis: An Introduction*. Springer, 2000.

R. E. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7:463–464, 1964.

Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

A Proofs

This appendix provides proofs for the closure, backpropagation, and fixed finite optimizer-update results of Sections 2–7. Constants hidden by $O(\cdot)$ are independent of the variable architecture and certificates. Dimension factors, gate counts, primitive costs, magnitude scales, and precision-amplification scales are displayed or bounded explicitly by polynomials in the padded input parameters. The operand-length bound includes c_κ, so exact certificate endpoints and rounded numerical values are both covered by the specified primitive costs.

A.1 Represented spaces, products, and uniform precision

We first spell out the representation-theoretic facts used implicitly in the paper.

Lemma 2 (Coordinate representations and full output)**.** *Coordinate-query and product-oracle names of a finite real tensor are polynomial-time interconvertible by query routing. If the discrete code also contains explicit-size padding bounding the number m of tensor coordinates, producing all m coordinates at a common precision is polynomial-time equivalent to answering one requested coordinate, uniformly in the padded input length.*

Proof. Route a query $(i, 1^n)$ to the ith component name, or obtain that component name by fixing i. For full materialization, make one coordinate query for each $i = 1, \dots, m$. Since $m \le \mathrm{poly}(|\kappa|)$, this multiplies a uniform second-order polynomial bound by a polynomial factor. Conversely, one can compute the full vector and select coordinate i. The same reasoning applies after flattening the tensor multi-index. Explicit-size padding supplies the uniform bound on the materialization cost. □

Lemma 3 (Finite sums and products over represented reals)**.** *Let $x_1, \dots, x_m$ and $y_1, \dots, y_m$ lie in a compact interval $[-B, B]$, where a dyadic upper bound $B \ge 1$ is part of the parameter. Then the maps*

$$(x_i)_{i=1}^m \mapsto \sum_{i=1}^m x_i, \qquad ((x_i, y_i))_{i=1}^m \mapsto \sum_{i=1}^m x_i y_i \tag{32}$$

are computable to error 2^{-n} in time $T_{\mathrm{io}} + \mathrm{poly}(m, n, r_B)$, where r_B is the supplied unary certificate scale for B and T_{io} charges finite-code decoding, oracle-query writing, oracle-answer processing, and output writing. The total running time is bounded by a second-order polynomial in the input-name length function and $n + |\kappa|$.

Proof. For the sum, set $p_s = n + \lceil \log_2 m \rceil + 2$. Query each x_i to precision $p_s + 2$, round the answer to $p_s + 1$ fractional bits, and add the resulting dyadics exactly. Each normalized input has error at most 2^{-p_s}, so the sum has error at most $m2^{-p_s} \le 2^{-n-2}$. Rounding the final sum to n fractional bits adds at most 2^{-n-1}, giving total error below 2^{-n}.

For products, set the working precision to

$$p = n + \lceil \log_2 m \rceil + \lceil \log_2(4B + 4) \rceil + 2. \tag{33}$$

Query each x_i, y_i to precision $p + 2$ and round each answer to $p + 1$ fractional bits. The normalized dyadics $\tilde{x}_i, \tilde{y}_i$ then approximate their inputs to error at most 2^{-p}, so

$$|x_i y_i - \tilde{x}_i \tilde{y}_i| \le |x_i|\,|y_i - \tilde{y}_i| + |\tilde{y}_i|\,|x_i - \tilde{x}_i| \le (2B + 1)2^{-p}, \tag{34}$$

using $|\tilde{y}_i| \le B + 1$. Summing over m terms gives error at most 2^{-n-1}. Compute the dyadic products and their sum exactly, then round the final result to n fractional bits; this adds at most 2^{-n-1}. For both maps, normalization gives operands and intermediate results of $O(n + \log m + r_B)$ bits, and the arithmetic work uses $O(m)$ operations. Finite-code decoding, query writing, answer processing and normalization, and output writing contribute T_{io}. Since there are $O(m)$ queries of polynomially bounded length, these costs satisfy the stated second-order polynomial bound. □

Lemma 4 (Precision propagation for certified programs)**.** *Let a straight-line program have A gates g_j, each L_j-Lipschitz on a certified dyadic argument box C_j containing its exact argument tuple on the promised input domain. Before each approximate evaluation, clip the argument tuple coordinatewise to C_j. Suppose input-coordinate error and each gate's combined evaluation/rounding error are at most δ. Put*

$$K_{\mathrm{prog}} = \prod_{j=1}^{A} (1 + L_j).$$

Then every output-coordinate error is at most $(A+1)K_{\mathrm{prog}}\delta$. In particular,

$$p = n + \left\lceil \log_2\bigl(2(A+1)K_{\mathrm{prog}}\bigr) \right\rceil, \qquad \delta = 2^{-p},$$

suffices for error at most 2^{-n-1}. The stored word length is $\max\{p + \lceil \log_2(2 + B_{\mathrm{prog}}) \rceil, c_\kappa\} + O(1)$, including the code lengths of exact certificate endpoints. If the gate count, summed log-amplification, magnitude scale, and evaluator costs are polynomially bounded in the encoded parameters, the operator is uniformly FP_2-computable.

Proof. Let E_j be the maximum error over all input and intermediate values available after gate j. Then $E_0 \le \delta$. Because the exact argument belongs to C_j, nonexpansiveness of coordinatewise clipping gives an argument error at most E_{j-1}. Thus

$$E_j \le \max\{E_{j-1}, L_j E_{j-1} + \delta\} \le (1 + L_j)E_{j-1} + \delta.$$

Writing $K_j = \prod_{i=1}^{j}(1 + L_i)$, induction gives $E_j \le (j+1)K_j\delta$: the induction step uses $K_j \ge 1$. The specified precision makes $E_A \le 2^{-n-1}$, leaving room for final output rounding. Gate evaluation and rounding can each receive a fixed number of additional guard bits so their combined local error is at most δ. This recurrence bounds input and internal errors together. If a separate input Lipschitz bound L_F is available, the larger choice based on $1 + L_F + (A+1)K_{\mathrm{prog}}$ is also sufficient and can be used to display both sources of conditioning. Algorithmically, choose integers b_j with $1 + L_j \le 2^{b_j}$ from the supplied scales and use $2^{\sum_j b_j}$ as an upper bound for K_{prog}. The required guard-bit count is then obtained by polynomial-size integer arithmetic on the scales. □

A.2 Smooth-primitive closure lemma

Proof of Lemma 1. Let A be the number of gates in the supplied program. Lemma 4 requires

$$p = n + O\left(\log(2 + A) + \sum_{j=1}^{A} \log(1 + L_j) \right).$$

The hypotheses make this polynomial in $n + |\kappa|$. The magnitude certificate makes total operand length polynomial as well. Executing the polynomial number of gates with their uniform evaluators, and charging all coordinate queries and output writes, therefore has a second-order polynomial time bound independent of the particular architecture.

For dense layers, the operations are additions, multiplications, and activation calls. On a gate box with magnitude at most B_{prog}, addition has Lipschitz constant at most 2 and multiplication at most $2B_{\mathrm{prog}}$ in the maximum norm. Activation gates use S_ℓ. The forward bounds A_ℓ, Z_ℓ and bounds for partial affine sums supply the ranges. Residual additions and linear maps are covered by the same gates. The scalar loss may either use its certified Lipschitz bound or be appended through its uniform polynomial precision modulus. A fixed finite number of such modulus-controlled additional primitives preserves a polynomial precision bound by composition. The displayed summed log-amplification bound supplies uniformity for variable program length. □

A.3 Forward, first-derivative, and second-derivative envelopes

Lemma 5 (Forward range envelope)**.** *For every layer ℓ, the activation vector satisfies $\|a^\ell\|_\infty \le A_\ell$ and the preactivation vector satisfies $\|z^\ell\|_\infty \le Z_\ell$.*

Proof. The assertion is true at layer 0 by the definition of A_0. Assuming $\|a^{\ell-1}\|_\infty \le A_{\ell-1}$, for every coordinate i,

$$|z_i^\ell| \le \sum_j |W_{ij}^\ell|\,|a_j^{\ell-1}| + |b_i^\ell| \le r_\ell A_{\ell-1} + \|b^\ell\|_\infty = Z_\ell. \tag{35}$$

Then $|a_i^\ell| = |\sigma_\ell(z_i^\ell)| \le A_\ell$ by definition of A_ℓ. □

We next construct the derivative envelopes used in the dense backpropagation bound. For scalar numerical variables ξ, η among the input coordinates, weights, biases, and represented label coordinates, define

$$e_\ell(\xi) := \left\| \frac{\partial a^\ell}{\partial \xi} \right\|_\infty, \qquad f_\ell(\xi,\eta) := \left\| \frac{\partial^2 a^\ell}{\partial \xi\, \partial \eta} \right\|_\infty.$$

Set $E_0 := 1$, $F_0 := 0$, and, for $1 \le \ell \le L$,

$$E_\ell := S_\ell\big(r_\ell E_{\ell-1} + A_{\ell-1} + 1\big),$$
$$F_\ell := T_\ell\big(r_\ell E_{\ell-1} + A_{\ell-1} + 1\big)^2 + S_\ell\big(r_\ell F_{\ell-1} + 2E_{\ell-1}\big).$$

The terms $A_{\ell-1}$ and 1 account for derivatives with respect to a current-layer weight or bias, respectively, so the recursions apply uniformly to all scalar numerical inputs. The following lemma establishes these envelopes and the resulting objective-Hessian bound.

Lemma 6 (First- and second-derivative envelopes)**.** *With $e_\ell, f_\ell, E_\ell, F_\ell$ defined above,*

$$e_\ell(\xi) \le E_\ell, \qquad f_\ell(\xi,\eta) \le F_\ell. \tag{36}$$

Consequently every second partial derivative of $J = \mathcal{L}(a^L, y)$ with respect to scalar numerical variables is bounded by

$$H_{\text{dense}} := V(E_L + 1)^2 + UF_L. \tag{37}$$

Proof. For $\ell = 0$, $a^0 = x$. The derivative of an input coordinate with respect to a scalar numerical variable is either 0 or 1, hence the bound $E_0 = 1$ is valid. Second derivatives of a^0 vanish, so $F_0 = 0$.

Assume the bounds hold at layer $\ell - 1$. Write

$$z_i^\ell = \sum_j W_{ij}^\ell a_j^{\ell-1} + b_i^\ell. \tag{38}$$

For a scalar variable ξ,

$$\partial_\xi z_i^\ell = \sum_j (\partial_\xi W_{ij}^\ell) a_j^{\ell-1} + \sum_j W_{ij}^\ell \partial_\xi a_j^{\ell-1} + \partial_\xi b_i^\ell. \tag{39}$$

The first term is either 0 or one activation coordinate, hence has absolute value at most $A_{\ell-1}$. The middle term is bounded by $r_\ell E_{\ell-1}$. The bias term is either 0 or 1. Therefore

$$|\partial_\xi z_i^\ell| \le r_\ell E_{\ell-1} + A_{\ell-1} + 1. \tag{40}$$

Since $a_i^\ell = \sigma_\ell(z_i^\ell)$,

$$|\partial_\xi a_i^\ell| = |\sigma_\ell'(z_i^\ell)|\,|\partial_\xi z_i^\ell| \le S_\ell(r_\ell E_{\ell-1} + A_{\ell-1} + 1) = E_\ell. \tag{41}$$

For second derivatives,

$$\partial^2_{\xi\eta} z_i^\ell = \sum_j W_{ij}^\ell \partial^2_{\xi\eta} a_j^{\ell-1} + \sum_j (\partial_\xi W_{ij}^\ell)\partial_\eta a_j^{\ell-1} + \sum_j (\partial_\eta W_{ij}^\ell)\partial_\xi a_j^{\ell-1}. \tag{42}$$

There is no second derivative of W^ℓ or b^ℓ because the coordinates are independent affine variables. Thus

$$|\partial^2_{\xi\eta} z_i^\ell| \le r_\ell F_{\ell-1} + 2E_{\ell-1}. \tag{43}$$

By the scalar chain rule,

$$\partial^2_{\xi\eta} a_i^\ell = \sigma_\ell''(z_i^\ell)(\partial_\xi z_i^\ell)(\partial_\eta z_i^\ell) + \sigma_\ell'(z_i^\ell)\partial^2_{\xi\eta} z_i^\ell. \tag{44}$$

Using the bounds above gives

$$|\partial^2_{\xi\eta} a^\ell_i| \le T_\ell(r_\ell E_{\ell-1} + A_{\ell-1} + 1)^2 + S_\ell(r_\ell F_{\ell-1} + 2E_{\ell-1}) = F_\ell. \tag{45}$$

This proves the recursion.

Finally, let $w = (a^L, y)$ and let $b_\xi = (\partial_\xi a^L, \partial_\xi y)$ be the derivative of the output-label pair with respect to ξ. Then $\|b_\xi\|_\infty \le E_L + 1$, and

$$\partial^2_{\xi\eta} J = b_\eta^\top \nabla^2_{ww} \mathcal{L}(w) b_\xi + \nabla_u \mathcal{L}(a^L, y)^\top \partial^2_{\xi\eta} a^L. \tag{46}$$

The chosen operator norms and envelopes imply

$$|\partial^2_{\xi\eta} J| \le \|b_\eta\|_\infty \|\nabla^2_{ww} \mathcal{L}(w) b_\xi\|_1 + U F_L \le V(E_L + 1)^2 + U F_L = H_{\text{dense}}. \tag{47}$$

□

On the certified common convex numerical-input domain, this bound controls the gradient's sensitivity to its original numerical inputs. The $\ell_\infty \to \ell_1$ loss-Hessian norm avoids an implicit output-dimension factor in the scalar bound, and the added 1 accounts for represented label coordinates. Errors introduced during the forward and reverse computations are controlled separately by the gate constants in Lemma 4.

Lemma 7 (Adjoint envelope)**.** *Define*

$$D_L := U S_L, \qquad D_\ell := S_\ell c_{\ell+1} D_{\ell+1} \quad (1 \le \ell < L). \tag{48}$$

Then

$$\|\delta^\ell\|_\infty \le D_\ell, \qquad \|\nabla_{W^\ell} J\|_\infty \le A_{\ell-1} D_\ell, \qquad \|\nabla_{b^\ell} J\|_\infty \le D_\ell. \tag{49}$$

Proof. For the final layer,

$$\|\delta^L\|_\infty \le \|\nabla_u \mathcal{L}(a^L, y)\|_1 \, \|\sigma'_L(z^L)\|_\infty \le U S_L. \tag{50}$$

If the claim holds at layer $\ell + 1$, then

$$\|\delta^\ell\|_\infty \le S_\ell \|(W^{\ell+1})^\top \delta^{\ell+1}\|_\infty \le S_\ell c_{\ell+1} \|\delta^{\ell+1}\|_\infty \le D_\ell. \tag{51}$$

The gradient bounds follow immediately from $\nabla_{W^\ell} J = \delta^\ell (a^{\ell-1})^\top$ and $\nabla_{b^\ell} J = \delta^\ell$, together with $\|a^{\ell-1}\|_\infty \le A_{\ell-1}$. □

A.4 Correctness of backpropagation

Proposition 1 (Backpropagation equations compute the gradient)**.** *For the smooth dense network of Section 3, the recurrence* (9)-(10) *and the formulas* (11) *compute* $\nabla_\theta J(\theta; x, y)$.

Proof. Let $\bar{a}^\ell := \partial J / \partial a^\ell$ and $\bar{z}^\ell := \partial J / \partial z^\ell$ denote row or column adjoints identified by the Euclidean inner product. At the output,

$$\bar{a}^L = \nabla_u \mathcal{L}(a^L, y), \qquad \bar{z}^L = \bar{a}^L \odot \sigma'_L(z^L), \tag{52}$$

which is exactly δ^L. For $\ell < L$, the dependence of J on a^ℓ through the next layer is

$$z^{\ell+1} = W^{\ell+1} a^\ell + b^{\ell+1}. \tag{53}$$

Thus

$$\bar{a}^\ell = (W^{\ell+1})^\top \bar{z}^{\ell+1}, \qquad \bar{z}^\ell = \bar{a}^\ell \odot \sigma'_\ell(z^\ell), \tag{54}$$

which gives (10). Finally, for a perturbation $(\Delta W^\ell, \Delta b^\ell)$ at layer ℓ,

$$\Delta z^\ell = \Delta W^\ell a^{\ell-1} + \Delta b^\ell. \tag{55}$$

The first variation of J is

$$\Delta J = \langle \delta^\ell, \Delta W^\ell a^{\ell-1} + \Delta b^\ell \rangle \tag{56}$$

$$= \langle \delta^\ell (a^{\ell-1})^\top, \Delta W^\ell \rangle + \langle \delta^\ell, \Delta b^\ell \rangle. \tag{57}$$

Hence the gradients are exactly those in (11). □

A.5 Arithmetic complexity of dense backpropagation

Proof of Theorem 1. For the forward pass at layer ℓ, each of the d_ℓ preactivation coordinates is a dot product of length $d_{\ell-1}$ plus one bias. Counting one multiplication and one addition per input coordinate gives $2d_\ell d_{\ell-1}$ scalar arithmetic operations; replacing the exact dot-product count $d_\ell d_{\ell-1} + d_\ell(d_{\ell-1} - 1) + d_\ell$ by $2d_\ell d_{\ell-1}$ changes only lower-order terms. Summing over layers gives $2s + O(\sum_\ell d_\ell)$, plus coordinatewise activation calls.

For the reverse pass, for each hidden layer $\ell < L$, the vector $(W^{\ell+1})^\top \delta^{\ell+1}$ has dimension d_ℓ and is formed by d_ℓ dot products of length $d_{\ell+1}$. This costs $2d_\ell d_{\ell+1} + O(d_\ell)$ operations. The Hadamard multiplication by $\sigma'_\ell(z^\ell)$ contributes $O(d_\ell)$ additional scalar operations. Summing gives

$$2\sum_{\ell=1}^{L-1} d_\ell d_{\ell+1} + O\left(\sum_{\ell=1}^{L-1} d_\ell\right). \tag{58}$$

The final-layer adjoint requires the loss-gradient primitive and $O(d_L)$ coordinatewise multiplications.

For the parameter gradients, $\nabla_{W^\ell} J = \delta^\ell (a^{\ell-1})^\top$ has $d_\ell d_{\ell-1}$ entries, each one scalar multiplication. Therefore materializing all weight gradients costs s scalar multiplications. The bias gradients are copies of δ^ℓ and cost at most $O(\sum_\ell d_\ell)$ to write. Combining these terms yields (12).

To obtain the bound $2\sum_{\ell=1}^{L-1} d_\ell d_{\ell+1} \le 2s$ under the usual dense feedforward accounting, observe that the term $d_\ell d_{\ell+1}$ is exactly the number of weights in layer $\ell + 1$, and hence

$$\sum_{\ell=1}^{L-1} d_\ell d_{\ell+1} = \sum_{j=2}^{L} d_{j-1} d_j \le s. \tag{59}$$

Thus the displayed implementation uses at most $5s + O(\sum_\ell d_\ell)$ scalar arithmetic operations, besides primitive calls. The full gradient has at least s scalar outputs. Consequently every algorithm that materializes it incurs $\Omega(s)$ scalar writes, and the implementation has tight $\Theta(s)$ cost in the arithmetic-and-output model. □

A.6 Bit complexity of dense backpropagation

Proof of Theorem 2. There are $q = N + d_0 + d_y$ varying scalar numerical inputs. For two numerical-input vectors u, v in the certified convex domain, write $\mathrm{BP}(u), \mathrm{BP}(v)$ for their full parameter gradients. The scalar second-partial bound gives

$$\|\,\mathrm{BP}(u) - \mathrm{BP}(v)\|_\infty \le q H_{\mathrm{dense}} \|u - v\|_\infty.$$

This estimates perturbations of the original inputs. For the actual finite-precision execution, use the gate argument boxes, Lipschitz constants, and magnitude bounds of the forward/reverse program. Clip each approximate argument tuple to its certified box. Lemma 4 gives total error at most

$$(A_{\mathrm{bp}} + 1) K_{\mathrm{bp}} 2^{-p}.$$

The choice in (16) makes this at most 2^{-n-1}; a final rounding of at most 2^{-n-1} gives the required accuracy. The gatewise bound controls amplification of errors in activation derivatives, products, adjoints, and partial sums.

To establish a uniform polynomial bound, the trace has polynomially many gates in the explicit architecture size. The envelopes bound activation and derivative values, and the adjoint recursion $D_L = US_L$, $D_\ell = S_\ell c_{\ell+1} D_{\ell+1}$ bounds all exact adjoints. The scale $\log_2(2 + D_\ell)$ is bounded by the sum of the scales $\log_2(2 + f)$ of its factors f, with a constant allowance. Partial sums are bounded by the number of terms times a product of the corresponding operand bounds, and gradient coordinates by $D_\ell A_{\ell-1}$. Thus polynomial logarithmic range bounds can be derived for the exact trace. Gate boxes and their full gate-image bounds are included in the program certificate. Addition, multiplication, activation, activation-derivative, and loss-gradient gates have respective bounds 2, $2B_{\mathrm{bp}}$, S_ℓ, T_ℓ, and V. Therefore $\log K_{\mathrm{bp}} = \sum_j \log(1 + L_j)$ is polynomially bounded in the padded input length. The recursions for E_ℓ, F_ℓ likewise give a polynomial logarithmic scale for H_{dense}.

Charge arithmetic at the total word length w, count every activation and activation-derivative call and the loss-gradient call, and include query writing, oracle-answer processing, certificate decoding, clipping, and output writing. This gives (17). The uniform primitive-evaluation assumptions then give one second-order polynomial runtime bound. □

A.7 Convolutional backpropagation

Proof of Theorem 3. Let p index an output position, r a kernel offset, c an input channel, and o an output channel. The discrete padding, stride, and dilation convention determines a partial index map $\pi(p, r)$ from output positions and offsets to input positions; invalid padded indices contribute zero. The preactivation is

$$z_{p,o} = b_o + \sum_{r=1}^{K} \sum_{c=1}^{C_{\mathrm{in}}} W_{r,c,o} X_{\pi(p,r),c}. \tag{60}$$

For each pair (p, o) this is a dot product of length KC_{in}, hence costs $2KC_{\mathrm{in}} + O(1)$ arithmetic operations. Since there are PC_{out} output coordinates, the forward affine cost satisfies

$$C_{\mathrm{fwd}}^{\mathrm{conv}} \le 2PKC_{\mathrm{in}}C_{\mathrm{out}} + O(PC_{\mathrm{out}}), \tag{61}$$

plus coordinatewise activation calls.

Let $\Delta_{p,o} = \partial J / \partial z_{p,o}$ be the preactivation adjoint. Differentiating the displayed formula gives

$$\frac{\partial J}{\partial W_{r,c,o}} = \sum_{p} \Delta_{p,o} X_{\pi(p,r),c}, \tag{62}$$

$$\frac{\partial J}{\partial b_o} = \sum_{p} \Delta_{p,o}, \tag{63}$$

$$\frac{\partial J}{\partial X_{q,c}} = \sum_{(p,r):\pi(p,r)=q} \sum_{o} \Delta_{p,o} W_{r,c,o}. \tag{64}$$

Equation (64) is the usual transposed convolution. It has the same number of active multiply-add incidences as the forward convolution, so the active arithmetic costs $2PKC_{\mathrm{in}}C_{\mathrm{out}} + O(PC_{\mathrm{out}})$. Materializing the full input adjoint also costs $O(IC_{\mathrm{in}})$ to initialize or write its coordinates. Equation (62) accumulates one product for every active quadruple (p, r, c, o) and therefore costs at most $2PKC_{\mathrm{in}}C_{\mathrm{out}}$ arithmetic operations, including additions into the accumulator. Writing the kernel and bias gradients adds $O(KC_{\mathrm{in}}C_{\mathrm{out}} + C_{\mathrm{out}})$, while bias-gradient accumulation costs $O(PC_{\mathrm{out}})$. Adding forward evaluation, input adjoints, and parameter-gradient accumulation yields

$$C_{\mathrm{bp}}^{\mathrm{conv}} \le 6PKC_{\mathrm{in}}C_{\mathrm{out}} + O(PC_{\mathrm{out}} + IC_{\mathrm{in}} + KC_{\mathrm{in}}C_{\mathrm{out}} + C_{\mathrm{out}}). \tag{65}$$

For the represented-real computation, the Lipschitz constant with respect to the original numerical inputs is at most $q_{\mathrm{conv}}H_{\mathrm{conv}}$ on the certified convex domain. The forward, input-adjoint, and shared-weight accumulation program has polynomially many gates in the displayed incidence and materialization counts. Its gate argument boxes and range bounds cover every partial sum and primitive call. Apply Lemma 4 with A_{conv} and K_{conv}; the precision in (24) gives the required output accuracy. Arithmetic is charged at the total word length determined by B_{conv}, with primitive and oracle/input/output costs added. The padded certificate scales make all these costs uniformly polynomial. □

A.8 Softmax, attention, and Transformer blocks

Lemma 8 (Softmax derivative envelope)**.** *For a score vector $s \in \mathbb{R}^m$ with $m \ge 1$, let $\rho = \mathrm{softmax}(s)$ and let δ_{ij} be the Kronecker delta, equal to 1 for $i = j$ and 0 otherwise. Then*

$$\frac{\partial \rho_i}{\partial s_j} = \rho_i(\delta_{ij} - \rho_j). \tag{66}$$

The first and second partial derivatives are bounded by universal constants on $\mathbb{R}^m$, independently of m. With m included in the explicit architecture size, rowwise softmax is uniformly FP_2-computable on bounded score boxes.

Proof. For one row, let

$$\rho_i = \frac{e^{s_i}}{\sum_k e^{s_k}}. \tag{67}$$

Differentiating gives

$$\partial_j \rho_i = \rho_i(\delta_{ij} - \rho_j). \tag{68}$$

Therefore

$$\sum_j |\partial_j \rho_i| = \rho_i(1-\rho_i) + \sum_{j \neq i} \rho_i \rho_j = 2\rho_i(1-\rho_i) \leq \frac{1}{2}. \tag{69}$$

This proves the ℓ_∞ Lipschitz bound. Differentiating once more gives

$$\partial^2_{jk} \rho_i = \rho_i\big[(\delta_{ij} - \rho_j)(\delta_{ik} - \rho_k) - \rho_j(\delta_{jk} - \rho_k)\big], \tag{70}$$

whose absolute value is bounded by a universal constant because every $\rho_i \in [0,1]$. Thus all second partial derivatives are uniformly bounded independently of the row length. For evaluation, compute a dyadic c satisfying $|c - \max_i s_i| \leq 1$ by approximating the m scores. The shifted formula with this c has denominator $\sum_k e^{s_k - c} \in [e^{-1}, me]$. Evaluation uses $O(m)$ primitive and arithmetic operations with $O(\log(2+m))$ additional guard bits for summation and normalization, giving uniform polynomial-time evaluation on bounded score boxes with explicitly encoded row length. □

Proof of Theorem 4. In the fused projection form of this implementation, the query, key, and value projections each multiply an $\ell \times d$ matrix by a $d \times d$ matrix. Each product costs $2\ell d^2$ scalar arithmetic operations up to lower-order terms, giving a combined cost of $6\ell d^2$. The output projection adds $2\ell d^2$.

For the attention scores, head a forms $Q_a K_a^\top$, an $\ell \times \ell$ matrix product with inner dimension d_k, costing $2\ell^2 d_k$. Summed over h heads, this is $2\ell^2 d$. Multiplication of the row-stochastic attention matrix by V_a costs another $2\ell^2 d_k$ per head, hence $2\ell^2 d$ overall. Scaling and rowwise softmax over all heads cost $O(h\ell^2)$ primitive operations. Therefore

$$C^{\mathrm{attn}}_{\mathrm{fwd}} \leq 8\ell d^2 + 4\ell^2 d + O(h\ell^2). \tag{71}$$

For reverse mode, each matrix multiplication $C = AB$ produces adjoints $\bar{A} = \bar{C}B^\top$ and $\bar{B} = A^\top \bar{C}$, two matrix multiplications of the same order as the forward multiplication. Including the saved forward computation, this gives a factor at most three for each linear-algebra block. For a saved softmax row ρ, the reverse pass applies the Jacobian-vector product

$$\bar{s}_i = \rho_i \left(\bar{\rho}_i - \sum_j \rho_j \bar{\rho}_j \right) \tag{72}$$

per row, costing linear time in the row length, hence $O(h\ell^2)$. Multiplying the forward bound by three on the matrix-product terms gives

$$C^{\mathrm{attn}}_{\mathrm{bp}} \leq 24\ell d^2 + 12\ell^2 d + O(h\ell^2). \tag{73}$$

For bit complexity, use the certified forward/reverse program. A softmax row may use the exact maximum as a represented-real value, since the maximum-value map is Lipschitz and computable even at ties. In the shifted formula the exponential arguments are nonpositive, every exponential is at most 1, and the denominator is at least 1. Exponentials below the requested error threshold are approximated by zero, giving polynomial-time evaluation at the requested absolute precision. The reverse softmax formula uses the saved probabilities and finite sums/products. The score, value, loss, adjoint, and partial-sum bounds in κ_{attn} therefore determine the gate Lipschitz and magnitude bounds. Lemma 4 yields (28); the total word length and primitive/input/output costs give a uniform polynomial bit bound. The final-objective Hessian bound supplies the original-input modulus, completing the sensitivity and execution-error estimates. □

For cross-attention, the query projections cost $2\ell_{\mathrm{q}} d^2$, key and value projections cost $4\ell_{\mathrm{kv}} d^2$, and the output projection costs $2\ell_{\mathrm{q}} d^2$, so the projection terms are bounded by $4(\ell_{\mathrm{q}} + \ell_{\mathrm{kv}})d^2$. The score and value products have $\ell_{\mathrm{q}}\ell_{\mathrm{kv}}$ query-key incidences and inner dimension summing to d across heads, giving $4\ell_{\mathrm{q}}\ell_{\mathrm{kv}} d$ forward cost and a factor-three reverse bound. This proves

$$C^{\mathrm{xattn}}_{\mathrm{bp}} \leq 12(\ell_{\mathrm{q}} + \ell_{\mathrm{kv}})d^2 + 12\ell_{\mathrm{q}}\ell_{\mathrm{kv}} d + O(h\ell_{\mathrm{q}}\ell_{\mathrm{kv}}). \tag{74}$$

For windowed attention, every query has at most r keys, so each ℓ^2 incidence count is replaced by $r\ell$, proving the stated windowed bound.

With μ, σ^2 defined in Section 6 and $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^d$, layer normalization with learned parameters $\alpha, \beta \in \mathbb{R}^d$ maps $x \in \mathbb{R}^d$ to

$$\alpha \odot \frac{x - \mu(x)\mathbf{1}}{\sqrt{\sigma^2(x) + \epsilon}} + \beta. \tag{75}$$

The mean and variance use finite sums and products, so they are polynomial-time computable. If $\epsilon > 0$, or if $\sigma^2(x) \geq \gamma > 0$ is certified, then reciprocal square root is evaluated on a compact interval separated from zero. Thus layer normalization is FP_2-computable, and adding residual connections and MLP sublayers preserves FP_2-computability by Lemma 1.

A.9 Fixed finite optimizer updates

Proof of Theorem 5. Each backpropagation call is uniformly FP_2 by Theorem 2. A fixed finite composition of these calls and the stipulated uniformly polynomial-time vector primitives remains FP_2. Their precision requirements are composed explicitly; a fixed number of polynomial precision moduli yields a polynomial modulus.

For the mini-batch mean, approximate each sample gradient to p_B fractional bits. The maximum-norm error of their average is at most 2^{-p_B}. The dyadic sum may be accumulated exactly at that fractional precision, with magnitude at most $B(G + 1)$. Division by the positive integer B and final rounding to error at most 2^{-n-2} leave total error below 2^{-n}. This costs $O(BNM(w_B))$ in addition to the sample-gradient evaluations. For the actual SGD update, if the approximations to $\theta, \eta, \bar{g}$ each have error at most $\delta \leq 1$, then before final arithmetic rounding the error is at most

$$\delta + \eta_{\max}\delta + (G + \delta)\delta.$$

The stated p_{upd} and fixed additional guard bits therefore suffice.

For square root, take $a, b \geq 0$ and assume $a \geq b$. Since

$$(\sqrt{a} - \sqrt{b})^2 \leq a - b,$$

an input error at most 2^{-2r-2}, after clipping to the nonnegative half-line, contributes at most 2^{-r-1} output error. The square root of a dyadic input can be computed with an additional absolute error of at most 2^{-r-1} by bisection or integer square-root computation in time polynomial in r and the magnitude scale. Thus the square-root precision modulus establishes polynomial-time computability at zero. In Adam, the denominator is at least $\epsilon > 0$, and lower bounds on $1 - \beta_i^t$ control the precision of bias correction. State and numerator bounds determine the precision of the remaining multiplication and division operations. This proves fixed finite RMSProp/Adam update computability. A polynomial bound on the composed precision modulus also gives uniform computability for input-dependent program lengths. □

B Additional Results

This appendix develops bounds for structured convolutions and Hessian-vector products, followed by concrete complexity illustrations and results on derivative selection, exact minimizer selection, and gradient-flow endpoints. The program bounds use the definitions and supporting lemmas in Appendix A.

B.1 Grouped and depthwise-separable convolutions

The incidence accounting in Appendix A.7 gives the following structured-convolution bounds. For grouped convolution with g groups, each output channel interacts with only C_{in}/g input channels, so every multiply-add incidence and kernel-coordinate count in that proof is divided by g, while the full input-adjoint materialization term remains $O(IC_{\mathrm{in}})$. Hence

$$C_{\mathrm{bp}}^{\mathrm{group}} \leq 6PK\frac{C_{\mathrm{in}}C_{\mathrm{out}}}{g} + O\left(PC_{\mathrm{out}} + IC_{\mathrm{in}} + K\frac{C_{\mathrm{in}}C_{\mathrm{out}}}{g} + C_{\mathrm{out}}\right). \tag{76}$$

For depthwise-separable convolution (Chollet, 2017; Howard et al., 2017), consisting of a depthwise spatial convolution followed by a $1 \times \cdots \times 1$ pointwise channel mixing layer, the depthwise

part has PKC_{in} incidences and the pointwise part has $PC_{\mathrm{in}}C_{\mathrm{out}}$ incidences. Applying the same forward/reverse/gradient factor and adding the full write terms gives

$$C_{\mathrm{bp}}^{\mathrm{sep}} \le 6P\big(KC_{\mathrm{in}} + C_{\mathrm{in}}C_{\mathrm{out}}\big) + O(P(C_{\mathrm{in}} + C_{\mathrm{out}}) + IC_{\mathrm{in}} + KC_{\mathrm{in}} + C_{\mathrm{in}}C_{\mathrm{out}} + C_{\mathrm{out}}). \quad (77)$$

The represented-real argument in Appendix A.7 applies to grouped and separable convolutions with their specified incidence counts and corresponding program certificates.

B.2 Hessian-vector products and iterative updates

For Hessian-vector products, let $u \in \mathbb{R}^q$ collect all varying numerical coordinates of (θ, x, y), let N be the number of parameters, let $v \in \mathbb{R}^N$ be the direction, and assume the required C^3 envelopes on a common convex domain. Suppose

$$|\partial_{\theta_i}\partial_{\theta_j} J| \le H_2, \qquad |\partial_{u_k}\partial_{\theta_i}\partial_{\theta_j} J| \le H_3, \qquad \|v\|_\infty \le R.$$

The joint map $(u, v) \mapsto \nabla^2_{\theta\theta} J(u)v$ then has maximum-norm Lipschitz bound

$$L_{\mathrm{HVP}} = qNH_3R + NH_2.$$

With uniformly polynomial-time evaluators for the derivatives used by the forward-over-reverse program (Pearlmutter, 1994; Schraudolph, 2002), let A_H, $K_H = \prod_j (1 + L_j)$, and B_H be its gate count, amplification bound, and magnitude bound. One sufficient choice is

$$p_H = n + \lceil \log_2(2[1 + L_{\mathrm{HVP}} + (A_H + 1)K_H]) \rceil,$$
$$w_H = \max\{p_H + \lceil \log_2(2 + B_H) \rceil, c_\kappa\} + O(1).$$

Let $A_{\mathrm{arith},H}$ be the program's arithmetic-gate count, $T_{\mathrm{io},H}$ its input/output cost, and $\mathcal{P}_H$ the list of all primitive calls, including additional derivative evaluations. The corresponding runtime is

$$T_{\mathrm{HVP}}(n) = O\left(T_{\mathrm{io},H} + A_{\mathrm{arith},H} M(w_H) + \sum_{g \in \mathcal{P}_H} T_g(p_H; \kappa, w_H) \right). \quad (78)$$

The arithmetic trace has constant-factor overhead relative to the differentiated gradient trace when each primitive's directional derivative has such an implementation; derivative-evaluator costs are counted explicitly.

To verify the Lipschitz bound, put $F_i(u, v) = \sum_{j=1}^N J_{\theta_i\theta_j}(u)v_j$. On the certified convex domain,

$$|\partial_{u_k} F_i| \le NH_3R, \qquad |\partial_{v_j} F_i| \le H_2.$$

Summing the $q + N$ input-coordinate sensitivities gives the maximum-norm Lipschitz bound $qNH_3R + NH_2$. This includes both dependence on the direction and dependence on every varying data or label coordinate. If only θ varies, $q = N$.

Forward differentiation of the smooth mathematical reverse program propagates primal/tangent pairs through its arithmetic and derivative primitives. Differentiation acts on the smooth primitive maps; the resulting primal/tangent program is then evaluated using the certified range projections and stable primitive evaluators. Under the stated primitive-derivative evaluator assumption, this is a finite program with the described arithmetic overhead. Apply Lemma 4 to its full gate trace, including the tangent values, with A_H, K_H, B_H. The displayed p_H is sufficient, and charging each additional derivative primitive separately gives (78).

For a fixed number of subsequent linear-algebra iterations, finite composition applies. For unary input 1^k, unroll all k iterations and apply the same program lemma to the entire trace, using the range and denominator bounds at every iterate. The conclusion is uniform when gate count, accumulated log-amplification, magnitude scales, and evaluator costs are polynomially bounded in $n + |\kappa| + k$.

B.3 Concrete complexity illustrations

We illustrate the arithmetic and precision bounds for the tractable training operators using concrete network architectures.

B.3.1 Numerical architecture examples

Consider a smooth dense network with widths $784, 128, 64, 10$. The numbers of weights and trainable parameters are

$$s = 784 \cdot 128 + 128 \cdot 64 + 64 \cdot 10 = 109{,}184,$$
$$N = s + 128 + 64 + 10 = 109{,}386.$$

With discrete labels, $q = N + d_0 = 110{,}170$. The displayed leading arithmetic term in (12) is

$$3s + 2(128 \cdot 64 + 64 \cdot 10) = 345{,}216.$$

Here $\sum_{\ell=1}^{L} d_\ell = 202$ in the lower-order term of (12), while the bound $5s$ gives a leading arithmetic term of $545{,}920$. The full bit runtime is given by (17).

Table 2 instantiates the leading arithmetic terms for the other training primitives. The convolution uses a 34×34 input with unit stride and no padding, producing 32×32 output positions with a 3×3 kernel. The attention example has $\ell = 128$, $d = 512$, and $h = 8$, so each head has dimension 64. The Transformer example uses this attention block and a positionwise feedforward subnetwork of widths $512, 2048, 512$.

Table 2: Numerical examples of the leading arithmetic terms. The first four rows include forward computation and reverse propagation for the respective network or block. The SGD row counts the final parameter update after the mini-batch gradient is available.

Operation	Configuration	Leading term
Dense	$784 \to 128 \to 64 \to 10$	345,216
Convolution	$P = 32^2$, $K = 9$, $C_{\text{in}} = 64$, $C_{\text{out}} = 128$	452,984,832
Self-attention	$\ell = 128$, $d = 512$, $h = 8$	905,969,664
Transformer	Same attention; $d_{\text{ff}} = 2048$	2,516,582,400
SGD update	$N = 109{,}386$	218,772

The convolutional entry is $6PKC_{\text{in}}C_{\text{out}}$ from (23). For attention, the projection and token-interaction terms in (27) are respectively $24\ell d^2 = 805{,}306{,}368$ and $12\ell^2 d = 100{,}663{,}296$. Each feedforward linear map in the Transformer contributes $6\ell d d_{\text{ff}}$ for forward computation and the two reverse matrix products, giving the combined leading expression

$$24\ell d^2 + 12\ell^2 d + 12\ell d d_{\text{ff}} = 2{,}516{,}582{,}400.$$

Finally, $\theta^+ = \theta - \eta\bar{g}$ uses one scalar multiplication and one subtraction per parameter, giving $2N = 218{,}772$ additional arithmetic operations; mini-batch gradient construction has the cost in (31).

B.3.2 Precision and attention-cost crossover

For a numerical precision example, consider a single-neuron network with a represented real label:

$$a = \tanh(\omega x + b), \qquad J(\theta; x, y) = \tfrac{1}{2}(a - y)^2,$$
$$\theta = (\omega, b), \qquad (\omega, b, x, y) \in [-1, 1]^4.$$

Here $N = 2$ and $q = 4$. Valid envelopes are $A_0 = Y_0 = r_1 = c_1 = A_1 = S_1 = 1$, $Z_1 = T_1 = U = 2$, and $V = 4$; Appendix A.3 gives $E_1 = 3$, $F_1 = 20$, and $H_{\text{dense}} = 104$.

Compute the gradient through eight gates: $t = \omega x$, $z = t + b$, $a = \tanh z$, $e = a - y$, $u = a^2$ (a unary square), $v = 1 - u$, $\delta = ev$, and $g_\omega = \delta x$, with $g_b = \delta$ an output copy. Certified argument boxes follow from $t, a \in [-1, 1]$, $z, e, \delta \in [-2, 2]$, and $u, v \in [0, 1]$, together with the input domain. The corresponding maximum-norm gate Lipschitz bounds are $(2, 2, 1, 2, 2, 1, 3, 3)$, with $A_{\text{bp}} = 8$ and $B_{\text{bp}} = 2$. Thus (16) yields

$$K_{\text{bp}} = 5184, \qquad C_{\text{bp}} = 1 + 4 \cdot 104 + 9 \cdot 5184 = 47073,$$
$$p = n + \lceil \log_2(2C_{\text{bp}}) \rceil = n + 17.$$

With gate arguments clipped to these boxes and input and combined local evaluation/rounding errors at most 2^{-p}, Lemma 4 gives gradient error at most 2^{-n} after final rounding to n fractional bits. For

$n = 20$, $p = 37$ fractional working bits are a uniform sufficient choice: seventeen fractional guard bits over the entire input box.

For attention, the ratio of the token-interaction term $12\ell^2 d$ to the projection term $24\ell d^2$ is $\ell/(2d)$. Thus the two terms coincide at $\ell = 2d$, which is 1024 for $d = 512$; the token-interaction term is larger for longer sequences.

B.4 Nonsmooth branch boundaries

We have remarked that nonsmooth derivative conventions are not globally computable as exact-real operators. We give a precise version here.

Proposition 2 (Discontinuous derivative selection is not computable)**.** *Let $g : \mathbb{R} \to \mathbb{R}$ be the derivative-selection map for ReLU with any convention satisfying $g(x) = 0$ for $x < 0$ and $g(x) = 1$ for $x > 0$. No single-valued extension of this map to $x = 0$ is computable as a represented real operator on all of $\mathbb{R}$.*

Proof. Every computable real function under the standard Cauchy representation is continuous. The function g has a jump discontinuity at 0, irrespective of the value assigned to $g(0)$: along $x_k = -2^{-k}$ the values tend to 0, while along $x_k = 2^{-k}$ the values tend to 1. Thus g is not continuous at 0 and cannot be computable.

Equivalently, assume a machine computes g, and run it on the all-zero Cauchy name of 0 with requested error less than $1/4$. It makes finitely many precision queries; let p_* be their maximum, taking $p_* = 0$ if there are none. Every ϵ with $0 < \epsilon < 2^{-p_*}$ admits positive and negative Cauchy names that answer zero to all those queries. If $g(0) \leq 1/2$, use ϵ, for which $g(\epsilon) = 1$; otherwise use $-\epsilon$, for which $g(-\epsilon) = 0$. The machine has the same transcript on the two inputs and cannot approximate both values to error less than $1/4$. □

Proposition 3 (Piecewise computability with encoded margins)**.** *Consider a family of polynomial-size programs with certified smooth gates and sign or finite-ranking decisions. Suppose every sign-test value has magnitude at least $\gamma > 0$, and every selected-versus-nonselected ranking gap is at least γ. Supply γ and its unary scale as in Definition 2, and uniform program certificates for all relevant branch-value and selected-branch computations. Then the derivative-selection operator for this family is uniformly FP_2-computable.*

Proof. Process decisions in execution order. At each decision, compute the relevant scalar values with certified error at most $\gamma/4$. The target accuracy uses the nonnegative precision request $\max\{0, \lceil \log_2(4/\gamma) \rceil\}$; the program precision bound determines the working accuracy for the preceding computation. The sign is then determined, and a selected-versus-nonselected difference changes by at most $\gamma/2$, so the selected support is determined as well. Subsequent decisions are evaluated on the already certified branch. There are polynomially many gates and decisions. The selected branch program satisfies the specified uniform range and precision hypotheses, so its derivative output can be evaluated in FP_2. The margin contributes its encoded precision scale in addition to the ordinary program-propagation costs. □

Proposition 3 gives polynomial-time derivative selection for hard attention, argmax outputs, and top-k routing on domains with positive gaps. We next establish the classifications of exact minimizer selection and gradient-flow endpoints.

B.5 Exact minimizer selection

Proposition 4 (Exact minimizer selection)**.** *Every single-valued exact-minimizer selector S on $C([0, 1])$ is noncomputable under evaluation names with supplied continuity moduli. Here $S(f) \in [0, 1]$ satisfies $f(S(f)) = \min_{t \in [0,1]} f(t)$. The affine family $f_a(t) = at$, $a \in [-1, 1]$, with a common Lipschitz bound establishes the result.*

Proof. The map $a \mapsto f_a$ with its common modulus is computable. The unique minimizer is 0 for $a > 0$ and 1 for $a < 0$. Whatever minimizer is selected when $a = 0$, the real map $a \mapsto S(f_a)$ is discontinuous at zero. Every computable real map under the Cauchy representation is continuous, a contradiction. □

B.6 Gradient-flow endpoint operators

The representation for this subsection supplies a point-evaluation oracle for $F = -\nabla V$ on a dyadic box $D \subseteq \mathbb{R}^d$, with unary precision requests. It also supplies the explicit dimension, an initial point $x_0 \in D$, a dyadic horizon $T \geq 0$, valid bounds $\|F\|_\infty \leq B$ and $\mathrm{Lip}_\infty(F) \leq L$ on D, and a promise that the exact trajectory stays in D through time T. A compatible continuity modulus is supplied or obtained from L. The discrete code includes unary 1^Λ for an integer $\Lambda \geq TL$, together with the padded magnitude scales and dyadic box codes. The lower-bound theorem supplies potential-value access together with gradient access.

Theorem 6 (Polynomial-space upper bound for certified flow endpoints)**.** *Under this representation, $E_T(V, x_0) = X(T)$ is $\mathrm{FPSPACE}_2$-computable.*

Proof. Let Π_D denote coordinatewise clipping to D. Normalize time to $[0, 1]$ and consider the globally defined field

$$G(x) = TF(\Pi_D x).$$

It is Λ-Lipschitz, has magnitude at most a supplied or computed $M \geq \max\{1, TB\}$, and its exact trajectory agrees with the promised original trajectory under time rescaling. The extended field gives a globally Lipschitz evaluator for the same endpoint.

Use rounded Euler iteration with step $h = 2^{-r}$ and $N = 2^r$ steps. Let e_j be the maximum-norm error of the numerical state after j steps relative to $X(Tjh)$. Take initial error $e_0 \leq \delta$, field-evaluation error at most δ, and rounding error at most ρ per step. The exact one-step truncation error is at most $\Lambda M h^2/2$. Consequently

$$e_{j+1} \leq (1 + h\Lambda)e_j + h\delta + \rho + \tfrac{1}{2}\Lambda M h^2,$$

and summing this recurrence gives

$$e_N \leq e^\Lambda \left(e_0 + \delta + N\rho + \tfrac{1}{2}\Lambda M h\right).$$

Choose

$$r = n + 2\Lambda + \lceil \log_2(2 + \Lambda M) \rceil + 8, \qquad \delta = 2^{-r}, \quad \rho = 2^{-2r}.$$

Since $e^\Lambda \leq 2^{2\Lambda}$, the last bound is less than 2^{-n}. Evaluating G to precision r requires only polynomially many query bits in r, the box code, and the scale of T. The current state has magnitude at most $\|x_0\|_\infty + M + 2$ and can be stored with $O(r)$ fractional bits. The step counter uses $r + 1$ bits. Reuse the current-state, oracle-query, answer, and arithmetic workspaces at each step. Since r is polynomial in $n + |\kappa|$ and oracle answer lengths enter through the name-length function, the total space is bounded by a second-order polynomial. This proves the upper bound. □

The conditioning parameter Λ controls the guard-bit count through the Gronwall factor e^Λ. The unary encoding makes this precision dependence polynomial in the input length, as in the Lipschitz-IVP framework of Kawamura and Cook (2012).

For a lower bound specific to gradient fields, fix

$$D = [-1, 3] \times [-1, 1], \qquad K = [-1/2, 5/2] \times [-1/2, 1/2].$$

Let $\mathcal{C}$ consist of potentials that are C^2 on a neighborhood of D, satisfy on D

$$|V| \leq 4, \qquad \|\nabla V\|_\infty \leq 2, \qquad \|\nabla^2 V\|_{\infty\to\infty} \leq 2,$$

and whose negative-gradient trajectory from $(0, 0)$ stays in K for $0 \leq t \leq 2$. Supply point-evaluation names of both V and ∇V, with the common modulus $p \mapsto p + 2$ and length-monotone padding. A name consists of these point-evaluation oracles and their common modulus; the displayed analytic and trajectory bounds define the represented domain.

Theorem 7 (A gradient-specific exponential query lower bound)**.** *On $\mathcal{C}$, the endpoint operator satisfies $E \in \mathrm{FPSPACE}_2 \setminus \mathrm{FP}_2$, where $E(V) = X(2)$. More precisely, a deterministic point-oracle algorithm computing it to maximum-norm error 2^{-n} has worst-case query count Q satisfying*

$$Q + 1 \geq \frac{2^{n/2}}{\sqrt{420}}.$$

The conclusion holds with both potential-value and gradient-evaluation access.

Proof. The upper bound is Theorem 6 with fixed data. For the lower bound define

$$\beta(u) = u^3(1-u)^3 \quad (0 \le u \le 1), \qquad \beta(u) = 0 \quad \text{otherwise.}$$

The function and its first two derivatives vanish at the endpoints, so its zero extension is C^2. Direct calculation gives

$$\int_0^1 \beta = \frac{1}{140}, \quad \|\beta\|_\infty \le \frac{1}{64}, \quad \|\beta'\|_\infty \le \frac{3}{16}, \quad \|\beta''\|_\infty \le \frac{3}{2}.$$

For example, with $a = u(1-u)$, the derivatives are $\beta' = 3a^2(1-2u)$ and $\beta'' = 6a(1-5a)$.

For a partition of $[0,1]$ into intervals $(a_i, a_i + h_i)$, define $g(s) = h_i^2 \beta((s-a_i)/h_i)$ on each open interval and zero at the joins and outside $[0,1]$. Then g is C^2 and has the same three uniform bounds as above. Set

$$V_g(s,z) = -s - zg(s).$$

Its gradient and Hessian are

$$\nabla V_g = (-1 - zg', -g), \qquad \nabla^2 V_g = \begin{pmatrix} -zg'' & -g' \\ -g' & 0 \end{pmatrix}.$$

Thus on D the required potential and gradient bounds hold, and the Hessian row-sum bound is at most $3/2 + 3/16 = 27/16 < 2$. The negative-gradient equations are

$$s' = 1 + zg'(s), \qquad z' = g(s), \qquad (s(0), z(0)) = (0,0).$$

For $t \le 2$, $0 \le z(t) \le 1/32$ and $|s'(t) - 1| \le 3/512$. The solution stays in K, s is increasing, and $s(2) > 1$. Writing $Z(s)$ for the second coordinate when the first coordinate is s gives

$$z(2) = \int_0^1 \frac{g(s)}{1 + Z(s)g'(s)}\, ds \ge \frac{2}{3} \int_0^1 g(s)\, ds.$$

These estimates also verify existence through the required horizon and membership in $\mathcal{C}$.

Run an endpoint algorithm on $V_0(s,z) = -s$ at precision n. Suppose it makes Q point queries, to either oracle. Take every queried first coordinate lying in $(0,1)$ and add the endpoints $0, 1$. The resulting partition has $r \le Q + 1$ positive gap lengths h_i summing to 1. Construct g on these gaps as above. At every queried first coordinate, $g = g' = 0$, so both V_g and its gradient agree exactly with V_0 and its gradient on every query. Their names can therefore share the complete finite transcript, including the common modulus and padding. The algorithm returns the same output.

Write $z_g(t)$ for the second trajectory coordinate under V_g. Convexity gives

$$z_g(2) \ge \frac{1}{210} \sum_{i=1}^{r} h_i^3 \ge \frac{1}{210 r^2} \ge \frac{1}{210(Q+1)^2}.$$

One output can be within 2^{-n} of both 0 and $z_g(2)$ only if this separation is at most 2^{1-n}, yielding the stated query bound. All names used can have the same linear length function: values are uniformly bounded, accuracy requests are unary, and answers are padded uniformly. On those names, every second-order polynomial time bound becomes an ordinary polynomial in n. The exponential query requirement therefore excludes FP_2 unconditionally. □

The fixed dimension, horizon, and derivative bounds yield an explicit time–space separation for gradient-flow endpoints. Potential and gradient evaluation provide the local information, and the distributed bump construction determines the number of point queries required to recover the endpoint.

C Additional related work

Polynomial-time reducibility relative to function oracles was studied by Mehlhorn (1976), and algebraic models of real computation are developed by Blum et al. (1998). Our represented-real analysis makes the costs of requested precision and oracle access explicit.

The sensitivity products in backpropagation are related to the analysis of long-term gradient propagation in recurrent networks by Bengio et al. (1994). Batch normalization (Ioffe and Szegedy, 2015)

uses finite sums, variances, and regularized reciprocal square roots, illustrating the same primitive operations that enter our certified-program analysis.

Momentum (Polyak, 1964) and accelerated-gradient methods (Nesterov, 1983) maintain additional iterate state. Diagonal AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman and Hinton, 2012) adapt coordinatewise scaling using accumulated or averaged squared gradients. Curvature-vector products and conjugate-gradient iterations support Hessian-free neural-network optimization (Martens, 2010). Our finite-update evaluation bounds complement convergence analyses of adaptive methods, including Adam and its variants (Reddi et al., 2018).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The claims match the results and assumptions in this paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: The paper states its smoothness, certificate, margin, and fixed-iteration assumptions, and identifies discontinuous selection, exact optimization, and flow solving as outside the local FP_2 regime.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes].

Justification: Assumptions are provided in full detail; complete proofs are included in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [N/A].

   Justification: The paper is theoretical and contains no experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [N/A].

Justification: No experiments, datasets, or experimental code are introduced.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [N/A].

   Justification: The paper contains no empirical training or test setting.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A].

   Justification: There are no experiments or statistical estimates.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [N/A].

   Justification: No experiments requiring compute resources are reported.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work is mathematical theory and uses no human subjects, private data, or deployed system.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: The contribution is foundational complexity theory with no data, model release, or direct deployment.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper releases no data or models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A].

    Justification: The paper uses no existing datasets, code assets, or models; prior scholarly work is cited in the bibliography.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A].

    Justification: The paper introduces no new dataset, code package, model, or benchmark asset.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: LLMs are not an important, original, or non-standard part of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.