Vanilla Neural Network - Derivations & Proofs

Paul F. Roysdon, Ph.D.

Contents

1 Mathematical Derivations & Proofs

1.1 Introduction

A *vanilla* (feedforward, fully connected) neural network (VNN) is a parametric function that composes affine transformations and elementwise nonlinearities to approximate a mapping from inputs to outputs. From first principles, its parameters are learned by minimizing a data-dependent empirical risk. The central computational tool is *backpropagation*, which is simply reverse-mode differentiation (chain rule) over the computation graph. We derive: (i) the forward model and its dimensions; (ii) empirical risk with common losses; (iii) complete backpropagation equations, including gradients with respect to weights and biases; (iv) specializations for regression and classification (softmax + cross-entropy); (v) regularization and optimization (SGD, momentum); and (vi) existence/expressivity remarks.

1.2 Data and Notation

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be i.i.d. samples with

$$\mathbf{x}_i \in \mathbb{R}^d \quad \text{(column vectors)}, \qquad y_i \in \mathbb{R} \text{ (regression) or } y_i \in \{1, \dots, K\} \text{ (classification)}.$$

For multiclass classification, encode labels as one-hot vectors $\mathbf{e}_{y_i} \in \{0,1\}^K$. A L-layer network has widths $m_0 = d$ (input), $m_1, \dots, m_{L-1}$ (hidden), and m_L (output). For layer $\ell = 1, \dots, L$,

$$\mathbf{W}^{(\ell)} \in \mathbb{R}^{m_\ell \times m_{\ell-1}}, \qquad \mathbf{b}^{(\ell)} \in \mathbb{R}^{m_\ell}.$$

We write elementwise (Hadamard) products as $\odot$, and apply nonlinearities elementwise.

1.3 Model Formulation (Forward Propagation)

Define activations $\mathbf{a}^{(0)} = \mathbf{x}$ and for $\ell = 1, \ldots, L$,

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)} \in \mathbb{R}^{m_\ell}, \tag{1}$$

$$\mathbf{a}^{(\ell)} = \phi^{(\ell)}\big(\mathbf{z}^{(\ell)}\big) \in \mathbb{R}^{m_\ell}. \tag{2}$$

Common hidden nonlinearities $\phi^{(\ell)}$: ReLU $(u)_+ = \max\{u, 0\}$, tanh, sigmoid $\sigma(u) = 1/(1 + e^{-u})$. Output layer depends on the task:

- **Regression:** $\phi^{(L)}$ is identity; prediction $\hat{y} = a^{(L)} \in \mathbb{R}$ (or $\mathbb{R}^{m_L}$).
- **Binary classification:** $m_L = 1$, $\phi^{(L)} = \sigma$, $\hat{p} = \sigma(z^{(L)})$.
- **Multiclass:** $m_L = K$, softmax $p_k(\mathbf{x}) = \dfrac{\exp(z_k^{(L)})}{\sum_{t=1}^{K} \exp(z_t^{(L)})}$.

Dimensions. $\mathbf{z}^{(\ell)}, \mathbf{a}^{(\ell)}, \mathbf{b}^{(\ell)} \in \mathbb{R}^{m_\ell}$; $\mathbf{W}^{(\ell)} \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$. The parameter set is $\boldsymbol{\theta} = \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}$.

1.4 Losses and Empirical Risk

Given prediction $\hat{\mathbf{y}}(\mathbf{x}; \boldsymbol{\theta})$ and target y, define per-sample loss $\ell(\hat{\mathbf{y}}, y)$ and empirical risk with optional ℓ_2 regularization:

$$\widehat{R}(\boldsymbol{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell\big(\hat{\mathbf{y}}(\mathbf{x}_i; \boldsymbol{\theta}),\, y_i\big) \;+\; \frac{\lambda}{2}\sum_{\ell=1}^{L} \|\mathbf{W}^{(\ell)}\|_F^2, \qquad \lambda \geq 0. \tag{3}$$

Typical choices:

$$\text{Squared error (regression): } \ell = \tfrac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2. \tag{4}$$

$$\text{Binary cross-entropy: } \ell = -\,y \log \hat{p} - (1 - y)\log(1 - \hat{p}). \tag{5}$$

$$\text{Multiclass cross-entropy: } \ell = -\sum_{k=1}^{K} \mathbf{e}_y(k)\, \log p_k. \tag{6}$$

1.5 Backpropagation: Full Derivation via Chain Rule

We derive gradients for Eqn. (3) w.r.t. $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$. Fix a single sample $(\mathbf{x}, \mathbf{y})$ and define the sample loss $J = \ell(\hat{\mathbf{y}}, \mathbf{y}) + \frac{\lambda}{2}\sum_\ell \|\mathbf{W}^{(\ell)}\|_F^2$. Define *error signals* (pre-activation sensitivities)

$$\boldsymbol{\delta}^{(\ell)} \;\triangleq\; \frac{\partial J}{\partial \mathbf{z}^{(\ell)}} \in \mathbb{R}^{m_\ell}. \tag{7}$$

By the chain rule and Eqn. (1),

$$\frac{\partial J}{\partial \mathbf{W}^{(\ell)}} = \frac{\partial J}{\partial \mathbf{z}^{(\ell)}}\,\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}\,\big(\mathbf{a}^{(\ell-1)}\big)^{\top} \;+\; \lambda\,\mathbf{W}^{(\ell)}, \tag{8}$$

$$\frac{\partial J}{\partial \mathbf{b}^{(\ell)}} = \frac{\partial J}{\partial \mathbf{z}^{(\ell)}}\,\frac{\partial \mathbf{z}^{(\ell)}}{\partial \mathbf{b}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}. \tag{9}$$

Thus backpropagation reduces to computing $\boldsymbol{\delta}^{(\ell)}$ for all layers.

Output layer sensitivities. Let $\mathbf{a}^{(L)}$ be the network output (after the output nonlinearity).

- **Squared error (identity output).** $J = \frac{1}{2}\|\mathbf{a}^{(L)} - \mathbf{y}\|^2 + \text{reg}$. Then $\dfrac{\partial J}{\partial \mathbf{a}^{(L)}} = \mathbf{a}^{(L)} - \mathbf{y}$ and

$$\boldsymbol{\delta}^{(L)} = \frac{\partial J}{\partial \mathbf{z}^{(L)}} = \left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right) \odot \phi^{(L)\prime}\big(\mathbf{z}^{(L)}\big) = \big(\mathbf{a}^{(L)} - \mathbf{y}\big) \odot \mathbf{1}. \tag{10}$$

  (Identity output $\Rightarrow \phi^{(L)\prime} \equiv \mathbf{1}$.)

- **Binary cross-entropy with sigmoid.** With $\hat{p} = \sigma(z^{(L)})$, $J = -y \log \hat{p} - (1-y)\log(1-\hat{p}) + \text{reg}$. A standard calculus identity yields

$$\boldsymbol{\delta}^{(L)} = \frac{\partial J}{\partial z^{(L)}} = \hat{p} - y. \tag{11}$$

  Proof. $\partial J/\partial \hat{p} = -\frac{y}{\hat{p}} + \frac{1-y}{1-\hat{p}}$ and $\partial \hat{p}/\partial z = \hat{p}(1-\hat{p})$; multiply to obtain $\hat{p} - y$. ∎

- **Multiclass cross-entropy with softmax.** With $\mathbf{p} = \text{softmax}(\mathbf{z}^{(L)})$, $J = -\sum_k \mathbf{e}_y(k) \log p_k + \text{reg}$. Then

$$\boldsymbol{\delta}^{(L)} = \mathbf{p} - \mathbf{e}_y. \tag{12}$$

  Proof. $\partial J/\partial \mathbf{z}^{(L)} = \mathbf{J}_{\text{softmax}}^{\top} \frac{\partial J}{\partial \mathbf{p}}$, where $\frac{\partial J}{\partial \mathbf{p}} = -\mathbf{e}_y \oslash \mathbf{p}$ and the softmax Jacobian $J_{kt} = p_k(\delta_{kt} - p_t)$. Multiplying simplifies to $\mathbf{p} - \mathbf{e}_y$. ∎

Hidden layer recursion. For $\ell = L-1, \ldots, 1$,

$$\boldsymbol{\delta}^{(\ell)} = \Big(\mathbf{W}^{(\ell+1)}\Big)^{\top} \boldsymbol{\delta}^{(\ell+1)} \odot \phi^{(\ell)\prime}\big(\mathbf{z}^{(\ell)}\big). \tag{13}$$

Proof. By chain rule, $\partial J/\partial \mathbf{z}^{(\ell)} = \Big(\partial J/\partial \mathbf{a}^{(\ell)}\Big) \odot \phi^{(\ell)\prime}(\mathbf{z}^{(\ell)})$, and $\partial J/\partial \mathbf{a}^{(\ell)} = \big(\mathbf{W}^{(\ell+1)}\big)^{\top} \boldsymbol{\delta}^{(\ell+1)}$ because $\mathbf{z}^{(\ell+1)} = \mathbf{W}^{(\ell+1)}\mathbf{a}^{(\ell)} + \mathbf{b}^{(\ell+1)}$. ∎

Per-sample gradients. Combine Eqns. (8)–(13) to obtain

$$\frac{\partial J}{\partial \mathbf{W}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}\big(\mathbf{a}^{(\ell-1)}\big)^{\top} + \lambda\, \mathbf{W}^{(\ell)}, \qquad \frac{\partial J}{\partial \mathbf{b}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}. \tag{14}$$

For a mini-batch $\mathcal{B}$, average (or sum) these per-sample gradients over $i \in \mathcal{B}$.

1.6 Optimization: Gradient Descent and Variants

Let $\eta > 0$ be a learning rate and $\widehat{\nabla}_{\mathcal{B}}$ the mini-batch gradient of $\widehat{R}$.

$$\textbf{SGD:}\quad \mathbf{W}^{(\ell)} \leftarrow \mathbf{W}^{(\ell)} - \eta\, \widehat{\nabla}_{\mathcal{B}} \mathbf{W}^{(\ell)}, \qquad \mathbf{b}^{(\ell)} \leftarrow \mathbf{b}^{(\ell)} - \eta\, \widehat{\nabla}_{\mathcal{B}} \mathbf{b}^{(\ell)}. \tag{15}$$

$$\textbf{Momentum:}\quad \mathbf{v}^{(\ell)} \leftarrow \beta\, \mathbf{v}^{(\ell)} + (1-\beta)\, \widehat{\nabla}_{\mathcal{B}} \mathbf{W}^{(\ell)}, \quad \mathbf{W}^{(\ell)} \leftarrow \mathbf{W}^{(\ell)} - \eta\, \mathbf{v}^{(\ell)}. \tag{16}$$

Adaptive methods (e.g., Adam) use per-parameter first/second-moment estimates; weight decay is the $\lambda\, \mathbf{W}^{(\ell)}$ term in Eqn. (14).

1.7 Initialization and Scaling (Practical Derivation)

To stabilize activations/gradients, choose i.i.d. entries with zero mean and layer-wise variance preserving transforms. For ReLU, He initialization: $W_{ij}^{(\ell)} \sim \mathcal{N}\big(0, \frac{2}{m_{\ell-1}}\big)$; for tanh/sigmoid, Xavier/Glorot: variance $\frac{2}{m_{\ell-1}+m_\ell}$. These follow by matching $\text{Var}(\mathbf{z}^{(\ell)})$ across layers under independence assumptions.

1.8 Expressivity (Remark)

With a non-polynomial activation (e.g., sigmoid, ReLU), a single hidden layer of sufficient width is a universal approximator of continuous functions on compact subsets of $\mathbb{R}^d$ (Cybenko/Hornik). Depth often yields more efficient representations, but training remains empirical risk minimization with gradients given above.

1.9 Algorithm (Vanilla Feedforward NN Training)

1. **Input:** data $\{(\mathbf{x}_i, y_i)\}$, architecture $\{m_\ell\}_{\ell=0}^{L}$, activations $\{\phi^{(\ell)}\}$, loss ℓ, regularization λ, optimizer hyperparameters.
2. **Initialize** $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}$ (e.g., He/Glorot).
3. **Repeat** for epochs and mini-batches $\mathcal{B}$:
   (a) **Forward:** for each $i \in \mathcal{B}$, compute Eqn. (1) up to $\mathbf{a}^{(L)}$ (and softmax if multiclass).
   (b) **Loss:** $J_i = \ell(\hat{\mathbf{y}}_i, y_i)$; accumulate $J_\mathcal{B} = \frac{1}{|\mathcal{B}|}\sum_i J_i + \frac{\lambda}{2}\sum_\ell \|\mathbf{W}^{(\ell)}\|_F^2$.
   (c) **Backward:** compute $\boldsymbol{\delta}^{(L)}$ via Eqn. (10), Eqn. (11), or Eqn. (12); then propagate Eqn. (13) down to $\ell = 1$.
   (d) **Gradients:** form $\partial J_\mathcal{B}/\partial \mathbf{W}^{(\ell)}$, $\partial J_\mathcal{B}/\partial \mathbf{b}^{(\ell)}$ by Eqn. (14).
   (e) **Update:** apply SGD/momentum/Adam steps.
4. **Output:** trained parameters $\widehat{\boldsymbol{\theta}}$; predictions via forward pass.

1.10 Detailed Proof of Weight-Gradient Formula

Proof. We justify $\frac{\partial J}{\partial \mathbf{W}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)}(\mathbf{a}^{(\ell-1)})^\top$ (ignoring decay). For neuron j in layer ℓ, $z_j^{(\ell)} = \sum_k W_{jk}^{(\ell)} a_k^{(\ell-1)} + b_j^{(\ell)}$. By definition $\delta_j^{(\ell)} = \partial J/\partial z_j^{(\ell)}$. Then

$$\frac{\partial J}{\partial W_{jk}^{(\ell)}} = \sum_r \frac{\partial J}{\partial z_r^{(\ell)}} \frac{\partial z_r^{(\ell)}}{\partial W_{jk}^{(\ell)}} = \frac{\partial J}{\partial z_j^{(\ell)}} \cdot a_k^{(\ell-1)} = \delta_j^{(\ell)}\, a_k^{(\ell-1)}.$$

Stacking indices (j, k) yields the outer product $\boldsymbol{\delta}^{(\ell)}(\mathbf{a}^{(\ell-1)})^\top$. ■

1.11 Summary of Variables and Their Dimensions

- $\mathbf{x}_i \in \mathbb{R}^d$: input vector; $y_i \in \mathbb{R}$ (regression) or $\{1, \dots, K\}$ (classification).
- n: number of samples; d: input dimension; K: # classes (if applicable).
- L: number of layers; m_ℓ: width of layer ℓ; $m_0 = d$; m_L output size.
- $\mathbf{W}^{(\ell)} \in \mathbb{R}^{m_\ell \times m_{\ell-1}}$, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{m_\ell}$: parameters.
- $\mathbf{z}^{(\ell)}, \mathbf{a}^{(\ell)} \in \mathbb{R}^{m_\ell}$: pre-activations and activations.
- $\phi^{(\ell)}$: elementwise nonlinearity at layer ℓ; ${\phi^{(\ell)}}'$ its derivative.
- $\boldsymbol{\delta}^{(\ell)} = \partial J/\partial \mathbf{z}^{(\ell)} \in \mathbb{R}^{m_\ell}$: backprop sensitivities.
- $\lambda \geq 0$: ℓ_2 weight-decay coefficient; $\eta > 0$: learning rate.

1.12 Summary

From first principles, a vanilla neural network composes affine maps and nonlinearities Eqn. (1) and learns parameters by minimizing empirical risk Eqn. (3). Backpropagation applies the chain rule to obtain: (i) output-layer errors (e.g., $\boldsymbol{\delta}^{(L)} = \mathbf{p} - \mathbf{e}_y$ for softmax–cross-entropy); (ii) hidden-layer recursion Eqn. (13); and (iii) weight/bias gradients as outer products Eqn. (14). Optimization proceeds by (stochastic) gradient methods with regularization and well-scaled initialization, yielding a flexible, universal function approximator.