Week 12: Neural Networks

Francisco Richter Ernst Wit

Introduction to Data Science (MSc)

1 Introduction to Neural Networks

Neural networks can be viewed as compositions of simple nonlinear functions. By stacking many layers of these simple transformations, one can approximate complex relationships between inputs and outputs. Learning occurs by adjusting parameters to minimize a loss measuring discrepancy between predictions and observed targets.

2 Foundations and Architecture

The Perceptron.

Definition 2.1 (Perceptron)**.** *A perceptron is a linear classifier that computes:*

$$\hat{y} = sign\left(\sum_{i=1}^{p} w_i x_i + b\right) = sign(\mathbf{w}^T\mathbf{x} + b)$$

where $\mathbf{w}$ *are weights,* b *is the bias, and* $sign(\cdot)$ *is the activation function.*

The perceptron learns a separating hyperplane by iteratively updating the weight vector when a point is misclassified. On linearly separable data, the updates converge to a separating solution. The following illustration shows a learned boundary on a toy two-dimensional dataset, emphasizing the geometry of linear separability.

Activation Functions.

Definition 2.2 (Activation Function)**.** *An activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ *introduces nonlinearity into the network. Common choices include:*

- ***Sigmoid****:* $\sigma(z) = \frac{1}{1+e^{-z}}$
- ***Hyperbolic tangent****:* $\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- ***ReLU****:* $\sigma(z) = \max(0, z)$
- ***Leaky ReLU****:* $\sigma(z) = \max(\alpha z, z)$ *where* $\alpha \in (0, 1)$

Universal approximation (informal).

A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy, provided the activation function is non-constant, bounded, and monotonically increasing.

Architecture (MLP).

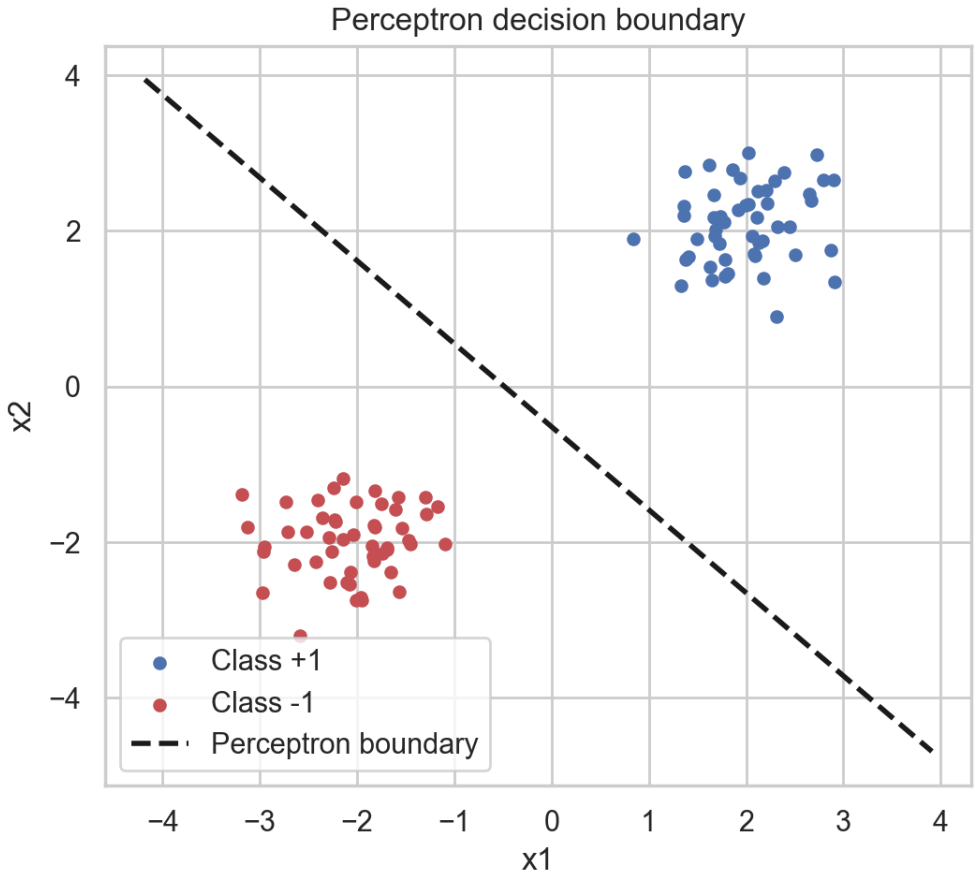

Figure 1: Perceptron decision boundary on a linearly separable dataset.

Definition 2.3 (Multilayer Perceptron (MLP)). *An MLP with L layers computes:*

$$\mathbf{a}^{(0)} = \mathbf{x} \quad \textit{(input layer)} \tag{1}$$
$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \quad \textit{(linear transformation)} \tag{2}$$
$$\mathbf{a}^{(l)} = \sigma^{(l)}(\mathbf{z}^{(l)}) \quad \textit{(activation)} \tag{3}$$

for $l = 1, 2, \ldots, L$, where $\mathbf{a}^{(L)}$ is the output.

Forward Propagation.
The forward pass computes the network output given input $\mathbf{x}$:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{a}^{(L)}$$

where $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$ are the network parameters.

3 Learning and Backpropagation

Regression Tasks.

Definition 3.1 (Mean Squared Error).

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})\|^2$$

Classification Tasks.

Definition 3.2 (Cross-Entropy Loss). *For binary classification:*

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

For multi-class classification:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(\hat{p}_{ik})$$

Backpropagation Algorithm.
Chain Rule Application.
Backpropagation uses the chain rule to compute gradients efficiently:
Backpropagation equations. For layer l:

$$\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}^{(L)}}\mathcal{L} \odot \sigma'^{(L)}(\mathbf{z}^{(L)}) \tag{4}$$
$$\boldsymbol{\delta}^{(l)} = [(\mathbf{W}^{(l+1)})^T\boldsymbol{\delta}^{(l+1)}] \odot \sigma'^{(l)}(\mathbf{z}^{(l)}) \tag{5}$$
$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)}(\mathbf{a}^{(l-1)})^T \tag{6}$$
$$\frac{\partial\mathcal{L}}{\partial\mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l)} \tag{7}$$

where $\odot$ denotes element-wise multiplication.

3.1 Computational Complexity

The computational complexity of backpropagation is $O(W)$ where W is the total number of weights, making it efficient for training large networks.

4 Optimization and Regularization

Gradient-based learning proceeds by repeated linearization of the objective and movement along the negative gradient direction. In practice, we compute gradients on mini-batches to reduce variance and improve hardware utilization. A simple visualization of a two-parameter loss surface and a gradient descent trajectory clarifies step sizes and curvature effects.

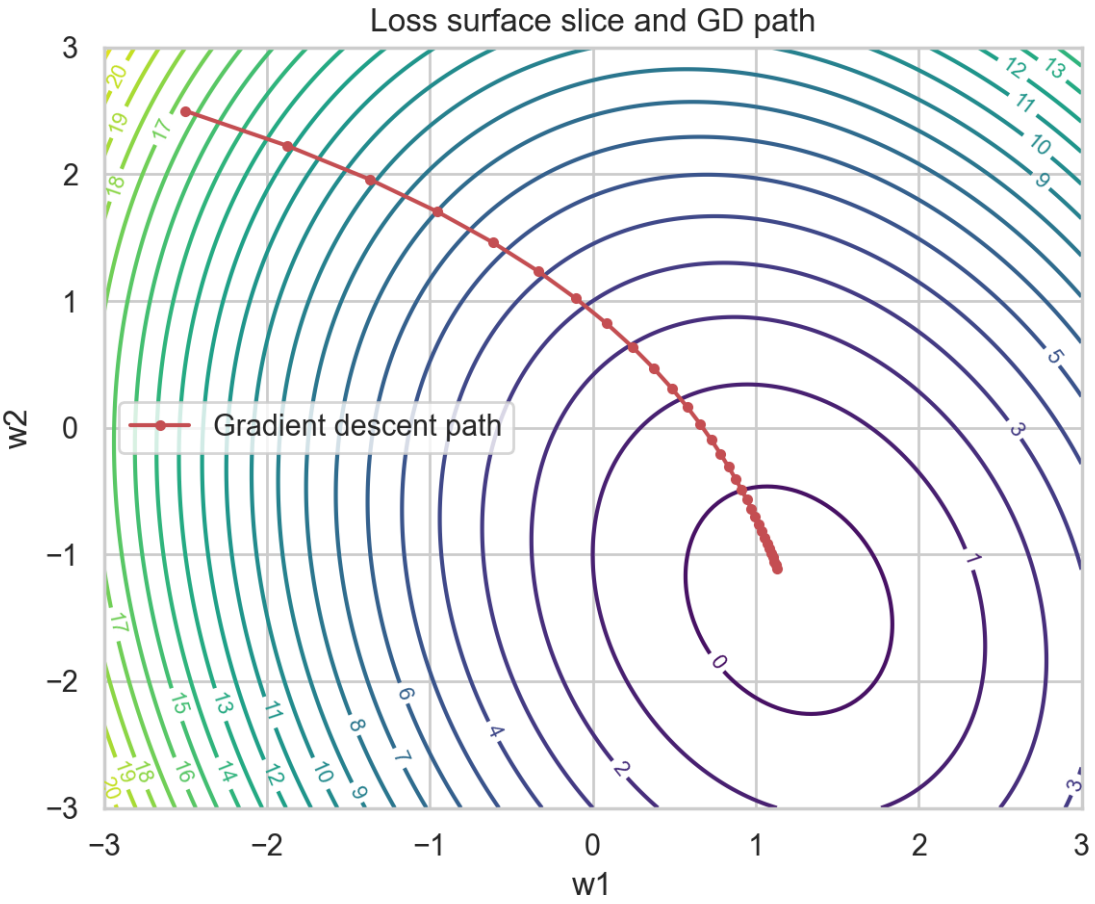

Figure 2: A loss surface slice with a gradient descent trajectory. Step sizes and curvature jointly determine progress.

Adaptive methods such as AdaGrad and Adam rescale or accumulate gradients to accommodate heterogeneity across parameters. This can accelerate early learning and stabilize training in the presence of ill-conditioned curvature, though careful tuning and validation remain essential.

Controlling generalization error requires explicit complexity control. Weight decay augments

the loss with an L2 penalty,

$$\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \lambda \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_F^2,$$

shrinking parameter norms toward zero. Dropout introduces multiplicative Bernoulli noise during training, implicitly averaging an ensemble of subnetworks. Batch normalization reduces internal covariate shift by standardizing layer inputs and learning affine corrections.

The effect of these techniques is readily seen on validation curves. As capacity grows or training continues, validation loss initially decreases and then increases. Early stopping selects the epoch that minimizes validation loss.

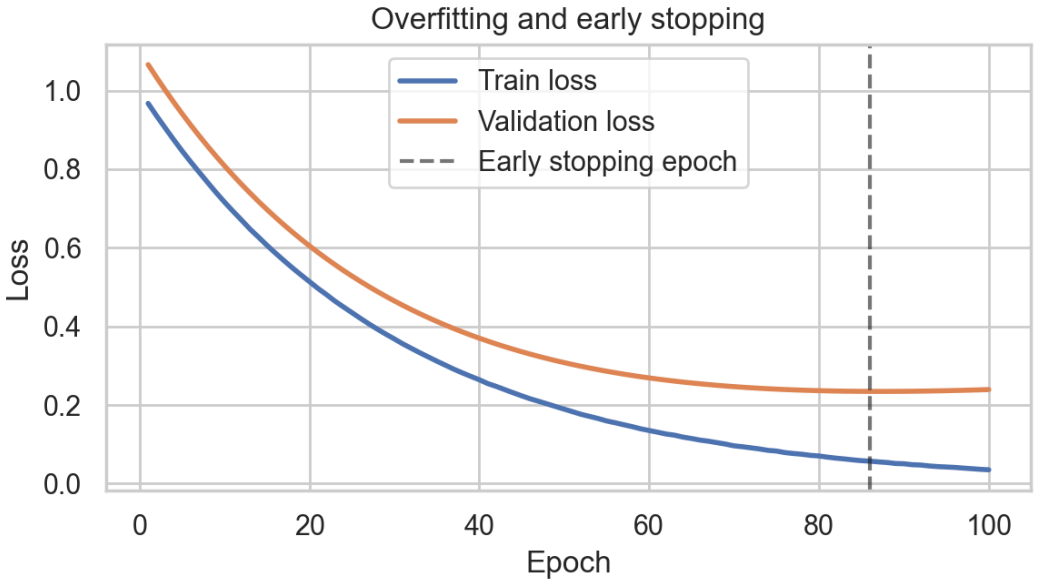

Figure 3: Training versus validation loss with early stopping at the optimal epoch.

Network Schematic.

To anchor notation, the following schematic depicts a minimal multilayer perceptron and its information flow from inputs, through a hidden layer with ReLU activation, to a sigmoid or softmax output. This diagram will be used to reference weight matrices and activations in the subsequent derivations.

5 Representation and Applications

The High-Dimensional Input Problem.

Definition 5.1 (High-Dimensional Classification Problem)**.** *Consider input data* $\mathbf{x} \in \mathbb{R}^d$ *where* d *is very large (e.g., image pixels* $d = 28 \times 28 = 784$ *for MNIST, or* $d = 32 \times 32 \times 3 = 3072$ *for CIFAR-10). The goal is to learn a mapping* $f : \mathbb{R}^d \to \{0, 1, \ldots, K-1\}$ *for classification.*

Example 5.1 (Image Recognition Challenge)**.** *For a* 4×4 *grayscale image (16 pixels), we have* $\mathbf{x} = (x_1, x_2, \ldots, x_{16})$ *where each* x_i *represents pixel intensity. The challenge: "Can we identify the letter* π *in a* 4×4 *grid?"*

This represents a typical machine learning question:

- ***Classification task**: Binary classification (π vs not π)*
- ***High-dimensional input**: 16-dimensional input space*
- ***Minimal semantic meaning**: Individual pixel values have little interpretable meaning*

Limitations of Linear Models.

Linear model limitations in high dimensions. *Setup.* Consider logistic regression for the image recognition problem:

$$Y_i = \begin{cases} 1 & \text{if image } i \text{ contains } \pi \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{X}_i = (X_{i1}, X_{i2}, \ldots, X_{i16}) \text{ (pixel values)}$$

The logistic regression model:

$$P(Y_i = 1|\mathbf{X}_i) = \frac{1}{1 + \exp(-\mathbf{X}_i^T \boldsymbol{\beta})}$$

This approach has fundamental limitations:

1. **Linear decision boundaries**: Cannot capture complex spatial patterns
2. **No spatial structure**: Treats pixels as independent features
3. **No translation invariance**: π in different positions treated as different patterns

Automatic Feature Learning.

Definition 5.2 (Representation Learning)**.** *Neural networks automatically learn hierarchical feature representations:*

$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \quad \textit{(low-level features)} \tag{8}$$

$$\mathbf{h}^{(2)} = \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \quad \textit{(mid-level features)} \tag{9}$$

$$\vdots \tag{10}$$

$$\mathbf{h}^{(L-1)} = \sigma(\mathbf{W}^{(L-1)}\mathbf{h}^{(L-2)} + \mathbf{b}^{(L-1)}) \quad \textit{(high-level features)} \tag{11}$$

$$\mathbf{y} = \textit{softmax}(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}) \quad \textit{(classification)} \tag{12}$$

In high-dimensional spaces, several phenomena occur that make traditional methods ineffective. In high-dimensional spaces, several phenomena occur that make traditional methods ineffective:

1. **Volume concentration**: In $\mathbb{R}^d$, as $d \to \infty$, the volume of a hypersphere concentrates near its surface.
2. **Distance concentration**: For random points $\mathbf{x}_1, \mathbf{x}_2$ in high dimensions:

$$\lim_{d\to\infty} \frac{\|\mathbf{x}_1 - \mathbf{x}_2\|_2}{\mathbb{E}[\|\mathbf{x}_1 - \mathbf{x}_2\|_2]} = 1$$

3. **Empty space phenomenon**: The ratio of the volume of a hypersphere to its enclosing hypercube approaches 0 as dimension increases.

Neural networks mitigate the curse of dimensionality through: Neural networks mitigate the curse of dimensionality through:

1. **Dimensionality reduction**: Each layer can reduce effective dimensionality by learning relevant features.
2. **Manifold learning**: Networks can learn that high-dimensional data lies on lower-dimensional manifolds.
3. **Hierarchical feature extraction**: Progressive abstraction reduces dependence on raw input dimensionality.

Universal Approximation and Representation Power.

Theorem 5.1 (Enhanced Universal Approximation)**.** *Let σ be a non-polynomial activation function. Then:*

1. ***Width sufficiency**: For any continuous function f on a compact set $K \subset \mathbb{R}^d$ and $\epsilon > 0$, there exists a neural network with one hidden layer of finite width that approximates f within ϵ.*
2. ***Depth efficiency**: For functions with compositional structure, deep networks require exponentially fewer parameters than shallow networks to achieve the same approximation accuracy.*

Proof. We prove the width sufficiency claim (density in $C(K)$) following Cybenko (1989) and Hornik (1991). Let $K \subset \mathbb{R}^d$ be compact and let $\mathcal{H}$ be the set of finite linear combinations of ridge functions $\sigma(\mathbf{w}^T\mathbf{x} + b)$ with $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$. Denote by $\overline{\mathcal{H}}$ the closure of $\mathcal{H}$ in the uniform norm on $C(K)$.

Assume, for contradiction, that there exists $f \in C(K)$ that is not in $\overline{\mathcal{H}}$. By the Hahn–Banach separation (or, equivalently, the Riesz representation theorem for duals of $C(K)$), there exists a nonzero finite signed Borel measure μ on K such that

$$\int_K g(\mathbf{x})\, d\mu(\mathbf{x}) = 0 \quad \text{for all } g \in \mathcal{H}, \qquad \text{but} \qquad \int_K f(\mathbf{x})\, d\mu(\mathbf{x}) \neq 0.$$

The first condition implies, in particular, that for all $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$,

$$\int_K \sigma(\mathbf{w}^T\mathbf{x} + b)\, d\mu(\mathbf{x}) = 0.$$

Define the pushforward measure $\nu_{\mathbf{w}}$ on $\mathbb{R}$ by $\nu_{\mathbf{w}}(B) = \mu\{\mathbf{x} \in K : \mathbf{w}^T\mathbf{x} \in B\}$ for Borel sets B. Then the above condition becomes

$$\int_{\mathbb{R}} \sigma(t + b)\, d\nu_{\mathbf{w}}(t) = 0 \quad \text{for all } b \in \mathbb{R},\ \mathbf{w} \in \mathbb{R}^d.$$

Since σ is non-polynomial, its translates span a set that separates finite signed measures (Cybenko, 1989). Hence the only finite signed measure on $\mathbb{R}$ that annihilates all translates $\{\sigma(\cdot + b) : b \in \mathbb{R}\}$ is the zero measure. Therefore $\nu_{\mathbf{w}} \equiv 0$ for every $\mathbf{w}$, which implies $\mu \equiv 0$ on K (because linear functionals $\mathbf{w}^T\mathbf{x}$ separate points on K). This contradicts the choice of nonzero μ. Thus $\overline{\mathcal{H}} = C(K)$, proving density and the width sufficiency claim.

For the depth efficiency claim, one constructs classes of compositional functions (e.g., parity-like or hierarchical multiplications) for which approximation by shallow networks requires exponentially many units, whereas a deep architecture reusing intermediate features attains polynomial size; see, for instance, Telgarsky (2016) and Eldan–Shamir (2016). This establishes the stated depth advantage. □

Theorem 5.2 (Depth vs Width Trade-off)**.** *There exist functions that can be represented by a deep network with polynomial width but require exponential width when represented by a shallow network.*

For sufficiently wide neural networks: For sufficiently wide neural networks:

- All local minima are global minima
- The number of saddle points grows exponentially with network size
- Gradient descent can escape saddle points efficiently

With probability at least $1-\delta$, With probability at least $1-\delta$,

$$\mathbb{E}[\mathcal{L}(\boldsymbol{\theta})] \leq \hat{\mathcal{L}}(\boldsymbol{\theta}) + \sqrt{\frac{KL(\rho||\pi) + \log(2\sqrt{n}/\delta)}{2n}}$$

where ρ is the posterior and π is the prior over parameters.

CNNs use:

Definition 5.3 (Convolution Operation)**.**

$$(\mathbf{f} * \mathbf{g})[i,j] = \sum_m \sum_n \mathbf{f}[m,n]\mathbf{g}[i-m, j-n]$$

CNNs use:

- **Local connectivity**: Neurons connect only to local regions
- **Parameter sharing**: Same weights used across spatial locations
- **Translation invariance**: Features detected regardless of position

RNNs process sequences via:

Definition 5.4 (RNN Update Equations)**.**

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y$$

LSTM addresses vanishing gradients through gating mechanisms:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad \text{(forget gate)} \tag{13}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad \text{(input gate)} \tag{14}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \quad \text{(candidate values)} \tag{15}$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \quad \text{(cell state)} \tag{16}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad \text{(output gate)} \tag{17}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t) \quad \text{(hidden state)} \tag{18}$$

In deep chains of nonlinearities, repeated multiplication by Jacobians may shrink or blow up gradient norms. A simple bound illustrates this behavior through products of layer operator norms and activation derivatives. Practice counters these issues by matching initialization to nonlinearity (e.g., Xavier for tanh, He for ReLU), introducing identity-like skip connections that shorten gradient

paths (residual networks), clipping gradients to cap extreme steps, and standardizing pre-activations within mini-batches using batch normalization. Each intervention targets a different part of the computation graph, collectively stabilizing learning dynamics.

Neural networks can overfit when model capacity outstrips sample size or when training proceeds past the point of optimal generalization. Early stopping halts optimization at the epoch that minimizes validation loss; weight decay biases parameters toward smaller norms; data augmentation injects invariances directly into the training distribution; cross-validation estimates out-of-sample error for robust hyperparameter choice.

As a case study, consider handwritten digit classification on grayscale 28×28 images. Each image is treated as a vector $\mathbf{x} \in \mathbb{R}^{784}$ after flattening, with label $y \in \{0, \dots, 9\}$. The goal is to learn a function mapping inputs to class probabilities that generalizes beyond the training set.

As a baseline, multinomial logistic (softmax) regression models class probabilities via linear scores $\mathbf{s}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ and $\mathrm{softmax}(\mathbf{s})$. Training minimizes empirical cross-entropy; the gradient equals predictions minus one-hot labels times inputs. Identifiability is ensured by fixing a reference class.

Gradient derivation. For a single example $(\mathbf{x}, y)$, let $\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and $\hat{\boldsymbol{\pi}} = \mathrm{softmax}(\mathbf{s})$. The per-example loss is $\ell = -\log \hat{\boldsymbol{\pi}}_y$. Using $\partial \ell / \partial s_k = \hat{\boldsymbol{\pi}}_k - \mathbf{1}\{k = y\}$, the chain rule gives

$$\nabla_{\mathbf{W}} \ell = (\hat{\boldsymbol{\pi}} - \mathbf{e}_y)\, \mathbf{x}^T, \qquad \nabla_{\mathbf{b}} \ell = \hat{\boldsymbol{\pi}} - \mathbf{e}_y,$$

and averaging over the sample yields the earlier expressions. The key step is the Jacobian of softmax, $\partial \hat{\boldsymbol{\pi}}_k / \partial s_j = \hat{\boldsymbol{\pi}}_k (\mathbf{1}\{j = k\} - \hat{\boldsymbol{\pi}}_j)$.

Numerical stability matters in practice: rather than computing $\log \sum_j \exp(s_j)$ directly, subtract the maximum logit $m = \max_j s_j$ and use $\log \sum_j \exp(s_j - m) + m$. This "log-sum-exp" trick avoids overflow while leaving softmax probabilities and the loss unchanged because adding a constant to all logits does not affect the distribution.

To increase expressivity, a one-hidden-layer MLP replaces linear scores with $\mathbf{s} = \mathbf{W}_2\, \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$, with ReLU activation $\sigma(z) = \max(0, z)$. Gradients follow from backpropagation: the output error is $\mathrm{softmax}(\mathbf{s}) - \mathbf{e}_y$, the hidden error gates with σ'.

Preprocessing rescales pixels to $[0, 1]$ (and optionally standardizes) to improve conditioning. A train/validation/test protocol supports principled model selection (hidden width, learning rate, weight decay) and final evaluation on held-out data. Early stopping and weight decay provide effective complexity control in practice.

Both approaches reduce effective capacity but via different mechanisms. Early stopping halts optimization when validation loss is minimized, implicitly constraining the trajectory length in parameter space; weight decay explicitly penalizes large weights, biasing the solution toward smaller-norm parameters and improving Hessian conditioning. Empirically, early stopping often preserves sharp discriminative features learned early, while L_2 shrinkage smooths decision boundaries. The training–validation curves above illustrate a typical selection of the optimal epoch for early stopping.