CS 395T: Foundations of Modern Generative AI Fall 2026

Lecture 1: Some History & Review of Deep Learning

Prof. Noah Golowich

1 Multi-layer Perceptrons

We begin with the simplest mathematical model of a neuron: a thresholded weighted sum. With some modifications, such "artificial neurons" form a key building block for a wide range of modern neural networks.

1.1 The MCP neural model

McCulloch and Pitts proposed an early mathematical model of neural activity [6]. They considered the setting of a neuron which has *excitatory* and *inhibitory* inputs. Let $m, r \in \mathbb{N}$ be the numbers of excitatory and inhibitory inputs. The excitatory inputs are $x_1, \dots, x_m \in \{0,1\}$, their weights are $w_1, \dots, w_m \in \mathbb{R}$, the inhibitory inputs are $z_1, \dots, z_r \in \{0,1\}$, and the threshold is $\vartheta \in \mathbb{R}$. The output $y \in \{0,1\}$ of the *McCulloch–Pitts (MCP) neuron* is

$$y = \begin{cases} 1, & \sum_{i=1}^{m} w_i x_i \geq \vartheta \quad \text{and} \quad z_j = 0 \text{ for every } j = 1, \dots, r, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Thus any active inhibitory input suppresses the neuron, while in the absence of inhibition the neuron fires exactly when the weighted excitatory input reaches the threshold. In this model the weights are fixed: they are chosen by the modeler, and no learning rule for adjusting them is part of the model.

1.2 The perceptron

Rosenblatt's perceptron uses the same mathematical prediction rule as the MCP model after the special inhibitory inputs are suppressed: it predicts by thresholding a weighted sum. For mathematical convenience, we also change notation slightly so that $y \in \{-1, 1\}$. In particular, Equation (1) is modified to

$$y = 2 \cdot \mathbf{1}\left[\sum_{i=1}^{m} w_i x_i \geq \vartheta\right] - 1. \tag{2}$$

The main conceptual difference is that Rosenblatt proposed that the weights should be *learned* from examples [8]. A nonzero threshold can be absorbed into the weight vector by appending to every input a coordinate equal to 1, so we work without loss of generality with threshold 0.

We now formalize a simple learning rule to actually learn the weights w_i, the *online perceptron algorithm* [1]. At time t, the algorithm receives an input $x_t \in \mathbb{R}^d$ and must predict its label $y_t \in \{-1, +1\}$. Its current weight vector is $w_t \in \mathbb{R}^d$. The algorithm is as follows.

1. Initialize $w_1 = 0$.
2. On input x_t, predict
$$\widehat{y}_t = \begin{cases} +1, & \langle w_t, x_t \rangle > 0, \\ -1, & \langle w_t, x_t \rangle \leq 0. \end{cases}$$

3. After observing y_t, update only if the prediction was wrong:

$$w_{t+1} = \begin{cases} w_t + y_t x_t, & \widehat{y}_t \neq y_t, \\ w_t, & \widehat{y}_t = y_t. \end{cases}$$

If an example is nonzero, scaling it to have Euclidean norm 1 does not change which side of a homogeneous separating hyperplane it lies on. We therefore state the mistake bound for normalized examples.

Theorem 1.1 (Perceptron mistake bound)**.** *Let $(x_t, y_t)_{t=1}^T$ be any sequence with $x_t \in \mathbb{R}^d$, $\|x_t\| = 1$, and $y_t \in \{-1, +1\}$. Suppose there is a unit vector $u \in \mathbb{R}^d$ for which*

$$\gamma := \min_{1 \le t \le T} y_t \langle u, x_t \rangle > 0.$$

Then the perceptron algorithm makes at most $1/\gamma^2$ mistakes on the sequence.

The quantity γ is the *margin*. Because the labels agree with the separator u, it is equivalently the minimum value of $|\langle u, x_t \rangle|$ when the examples have unit norm.

Proof. Let M be the total number of mistakes. If $M = 0$, the claimed bound is immediate, so assume $M \ge 1$. Write $(\widetilde{x}_s, \widetilde{y}_s)$ for the example and label on which the s-th mistake occurs, and let v_s be the weight vector immediately after that mistake. Thus $v_0 = 0$ and

$$v_s = v_{s-1} + \widetilde{y}_s \widetilde{x}_s, \qquad s = 1, \dots, M. \tag{3}$$

We first measure progress in the direction of the target separator u. Taking the inner product of (3) with u gives

$$\langle v_s, u \rangle = \langle v_{s-1}, u \rangle + \widetilde{y}_s \langle \widetilde{x}_s, u \rangle \ge \langle v_{s-1}, u \rangle + \gamma,$$

where the inequality is the definition of the margin. Starting from $\langle v_0, u \rangle = 0$ and applying this inequality once for each of the M mistakes yields

$$\langle v_M, u \rangle \ge M\gamma. \tag{4}$$

We next bound the length of the learned weight vector. Expanding the squared norm in (3),

$$\|v_s\|^2 = \|v_{s-1}\|^2 + 2\widetilde{y}_s \langle v_{s-1}, \widetilde{x}_s \rangle + \|\widetilde{x}_s\|^2 .$$

The update occurs only after a mistake. If $\widetilde{y}_s = +1$, a mistake means $\langle v_{s-1}, \widetilde{x}_s \rangle \le 0$; if $\widetilde{y}_s = -1$, a mistake means $\langle v_{s-1}, \widetilde{x}_s \rangle > 0$. In both cases, $\widetilde{y}_s \langle v_{s-1}, \widetilde{x}_s \rangle \le 0$. Since $\|\widetilde{x}_s\| = 1$, we obtain

$$\|v_s\|^2 \le \|v_{s-1}\|^2 + 1.$$

The base case is $\|v_0\|^2 = 0$. Inductively applying the preceding inequality for $s = 1, \dots, M$ therefore gives

$$\|v_M\|^2 \le M, \qquad \text{and hence} \qquad \|v_M\| \le \sqrt{M}. \tag{5}$$

Finally, the Cauchy–Schwarz inequality and $\|u\| = 1$ imply $\langle v_M, u \rangle \le \|v_M\|$. Combining this with (4) and (5) yields

$$M\gamma \le \sqrt{M}.$$

Because $M > 0$, dividing by $\sqrt{M}$ gives $\sqrt{M}\gamma \le 1$, and squaring gives $M \le 1/\gamma^2$, as claimed. □

1.3 Multi-layer perceptrons

A *multi-layer perceptron (MLP)* may be viewed as a generalization of a perceptron which collects many perceptrons, groups them into layers, and then stacks them on top of each other. More formally, it is a feedforward composition of affine maps and coordinatewise nonlinearities. Fix an input dimension $d \in \mathbb{N}$, a number $L \in \mathbb{N}$ of hidden layers, and layer widths $n_1, \dots, n_L \in \mathbb{N}$; set $n_0 = d$. For an input $x \in \mathbb{R}^d$, define $h^{(0)}(x) = x$. For each hidden layer $\ell = 1, \dots, L$, choose a matrix $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, a bias $b^{(\ell)} \in \mathbb{R}^{n_\ell}$, and a scalar activation function $\sigma_\ell : \mathbb{R} \to \mathbb{R}$, and set

$$z^{(\ell)}(x) = W^{(\ell)} h^{(\ell-1)}(x) + b^{(\ell)},$$
$$h^{(\ell)}(x) = \sigma_\ell(z^{(\ell)}(x)),$$

where σ_ℓ is applied to each coordinate. A scalar-output MLP has the form

$$f_\theta(x) = a^\top h^{(L)}(x) + c, \tag{6}$$

where $a \in \mathbb{R}^{n_L}$ and $c \in \mathbb{R}$. The symbol θ denotes the collection of all matrices, biases, and output weights. Threshold activations recover networks of perceptrons; the activation used most often below is the *rectified linear unit* (ReLU).

2 Expressivity

In this section, we consider how powerful MLPs are, via the lens of expressivity.

2.1 Representing Boolean functions

Theorem 2.1 (Boolean universality)**.** *Every function $F : \{0,1\}^d \to \{0,1\}$ is represented exactly by an MLP with one hidden layer and threshold activations.*

Proof. For every $a \in \{0,1\}^d$ with $F(a) = 1$, one hidden unit computes the indicator $x \mapsto \mathbf{1}[x = a]$ by thresholding $\sum_{i:a_i=1} x_i + \sum_{i:a_i=0}(1 - x_i)$ at $d - \frac{1}{2}$. An output threshold unit takes the OR of these minterms by thresholding their sum at $\frac{1}{2}$; when F is identically zero, use the constant-zero output. □

This is the disjunctive-normal-form construction: an OR of ANDs of literals. Its width can be as large as the number of satisfying inputs, so universality alone does not imply an efficient representation.

2.2 Approximating continuous functions

Define the ReLU activation $\rho : \mathbb{R} \to \mathbb{R}$ by

$$\rho(t) = \mathrm{ReLU}(t) := \max\{0, t\}.$$

For a compact set $K \subseteq \mathbb{R}^d$, let $C(K)$ be the vector space of continuous functions $f : K \to \mathbb{R}$, equipped with the uniform norm $\|f\|_\infty := \sup_{x \in K} |f(x)|$.

Theorem 2.2 (Universal approximation by ReLU; Cybenko [3])**.** *Let $K \subseteq \mathbb{R}^d$ be compact. For every $f \in C(K)$ and every $\epsilon > 0$, there are an integer $N \in \mathbb{N}$, output weights $a_1, \dots, a_N \in \mathbb{R}$, input weights $w_1, \dots, w_N \in \mathbb{R}^d$, and biases $b_1, \dots, b_N \in \mathbb{R}$ such that*

$$\sup_{x \in K} \left| f(x) - \sum_{j=1}^{N} a_j \rho(\langle w_j, x \rangle + b_j) \right| < \epsilon.$$

Proof. Let $\mathcal{H}_K \subseteq C(K)$ be the linear span of all ridge functions

$$x \longmapsto \rho\big(\langle w, x\rangle + b\big), \qquad w \in \mathbb{R}^d, \quad b \in \mathbb{R}.$$

An element of $\mathcal{H}_K$ is exactly a finite-width, one-hidden-layer ReLU network. We prove that the closure of $\mathcal{H}_K$ in the uniform norm is all of $C(K)$.

Suppose for contradiction that the closed linear subspace $\overline{\mathcal{H}_K}$ is a proper subset of $C(K)$. Choose $f_0 \in C(K) \setminus \overline{\mathcal{H}_K}$. The Hahn–Banach theorem in the separation form stated in Theorem A.1 gives a continuous linear functional $\Lambda : C(K) \to \mathbb{R}$ such that

$$\Lambda(h) = 0 \quad \text{for every } h \in \overline{\mathcal{H}_K}, \qquad \text{but} \qquad \Lambda(f_0) = 1.$$

By the Riesz representation theorem, stated in Theorem A.2, there is a finite signed regular Borel measure μ on K such that

$$\Lambda(g) = \int_K g(x)\, d\mu(x) \qquad \text{for every } g \in C(K).$$

The measure μ is nonzero because $\Lambda(f_0) = 1$. On the other hand, every ReLU ridge function belongs to $\mathcal{H}_K$, so

$$\int_K \rho\big(\langle w, x\rangle + b\big)\, d\mu(x) = 0 \qquad \text{for every } w \in \mathbb{R}^d,\ b \in \mathbb{R}. \tag{7}$$

We will show directly that (7) forces $\mu = 0$, giving the desired contradiction.

Fix $w \in \mathbb{R}^d$ and define $R_w := \max_{x \in K} |\langle w, x\rangle|$, which is finite because K is compact. Let ν_w be the pushforward of μ under the map $x \mapsto \langle w, x\rangle$: for every Borel set $B \subseteq [-R_w, R_w]$, set

$$\nu_w(B) := \mu\big(\{x \in K : \langle w, x\rangle \in B\}\big).$$

Equation (7) says that

$$\int_{-R_w}^{R_w} \rho(t + b)\, d\nu_w(t) = 0 \qquad \text{for every } b \in \mathbb{R}. \tag{8}$$

The constant function belongs to $\mathcal{H}_K$, since $1 = \rho(1)$, and the identity satisfies $t = \rho(t) - \rho(-t)$. Consequently, ν_w integrates both the constant function and the identity function to zero. Every continuous piecewise-linear function q on $[-R_w, R_w]$ can be written as

$$q(t) = \alpha + \beta t + \sum_{j=1}^{s} c_j \rho(t - r_j)$$

for some integer $s \geq 0$, coefficients $\alpha, \beta, c_1, \ldots, c_s \in \mathbb{R}$, and breakpoints $r_1, \ldots, r_s \in [-R_w, R_w]$. Equation (8) therefore implies $\int q\, d\nu_w = 0$ for every such q. Continuous piecewise-linear functions are uniformly dense in $C([-R_w, R_w])$: one may linearly interpolate any continuous function on an increasingly fine partition. Since ν_w is finite, uniform convergence $q_n \to q$ implies

$$\left| \int (q_n - q)\, d\nu_w \right| \leq \|q_n - q\|_\infty\, |\nu_w|([-R_w, R_w]) \longrightarrow 0,$$

where $|\nu_w|$ is the total-variation measure. Hence ν_w integrates every continuous function to zero, and the uniqueness part of the Riesz representation theorem gives $\nu_w = 0$.

This conclusion holds for every $w \in \mathbb{R}^d$. In particular, for every $\xi \in \mathbb{R}^d$, the Fourier transform of μ satisfies

$$\widehat{\mu}(\xi) := \int_K e^{\mathrm{i}\langle \xi, x\rangle}\, d\mu(x) = \int_{-R_\xi}^{R_\xi} e^{\mathrm{i}t}\, d\nu_\xi(t) = 0.$$

The uniqueness theorem for Fourier transforms of finite signed measures now gives $\mu = 0$. One way to see this final uniqueness step is that the complex linear span of the functions $x \mapsto e^{\mathrm{i}\langle \xi, x\rangle}$ contains constants, is closed under products and complex conjugation, and separates points of K; the Stone–Weierstrass theorem makes this span uniformly dense in $C(K)$. Thus $\mu = 0$, contradicting $\Lambda(f_0) = 1$. We conclude that $\overline{\mathcal{H}_K} = C(K)$, which is precisely the claimed approximation statement. $\square$

The preceding theorem gives no quantitative control on the necessary width N. To state one representative rate, we introduce the relevant smoothness class. A *multi-index* $\alpha = (\alpha_1, \dots, \alpha_d) \in (\mathbb{N} \cup \{0\})^d$ has order $|\alpha|_1 := \sum_{j=1}^d \alpha_j$, and $D^\alpha f$ denotes the corresponding weak partial derivative. For $n \in \mathbb{N}$ and $p \in [1, \infty]$, the Sobolev space $W^{n,p}([0,1]^d)$ consists of functions for which $D^\alpha f \in L^p([0,1]^d)$ whenever $|\alpha|_1 \le n$. We use the norm

$$\|f\|_{W^{n,p}} := \begin{cases} \left(\displaystyle\sum_{|\alpha|_1 \le n} \|D^\alpha f\|_{L^p}^p \right)^{1/p}, & p < \infty, \\ \displaystyle\max_{|\alpha|_1 \le n} \|D^\alpha f\|_{L^\infty}, & p = \infty, \end{cases}$$

and let $\mathcal{F}_{n,d}^p := \{f : \|f\|_{W^{n,p}} \le 1\}$ be its unit ball.

Theorem 2.3 (Smooth-function approximation rate; Mhaskar [7])**.** *Let $n \in \mathbb{N}$ and $p \in [1, \infty]$. Suppose an activation $\sigma : \mathbb{R} \to \mathbb{R}$ is infinitely differentiable on an open interval $I \subseteq \mathbb{R}$ and there is a point $x_0 \in I$ such that $\sigma^{(r)}(x_0) \neq 0$ for every $r \in \mathbb{N} \cup \{0\}$. Then there is a constant $C > 0$, depending only on d, n, p, and σ, with the following property: for every $f \in \mathcal{F}_{n,d}^p$ and every $\epsilon > 0$, there is a shallow network*

$$g(x) = \sum_{j=1}^N a_j \sigma(\langle w_j, x\rangle + b_j)$$

such that $\|f - g\|_{L^p([0,1]^d)} \le \epsilon$ and $N \le C\epsilon^{-d/n}$.

Unlike Theorem 2.2, it provides an approximation rate, but the factor $\epsilon^{-d/n}$ suffers from the curse of dimensionality: the required width grows exponentially with the dimension d. This is unavoidable in the worst case [5].

2.3 Benefits of depth

Next, we show that utilizing *deeper* networks can (sometime) help, from an expressivity standpoint. For this statement, we used a biased version of a ReLU gate, which maps its vector of incoming values v to $\rho(\langle a, v\rangle + b)$ for some weight vector a and bias b. The number of *distinct parameters* of a network counts the different scalar values used among all weights and biases, allowing the same value to be reused at many gates.

Theorem 2.4 (Depth separation for ReLU networks; Telgarsky [9])**.** *For every pair of integers $k, d \ge 1$, there is a function $f : \mathbb{R}^d \to \mathbb{R}$ computed by a ReLU network with $2k^3 + 8$ layers, $3k^3 + 12$ total nodes, and $4 + d$ distinct parameters such that every ReLU network $g : \mathbb{R}^d \to \mathbb{R}$ with at most k layers and at most 2^k nodes satisfies*

$$\int_{[0,1]^d} |f(x) - g(x)|\, dx \ge \frac{1}{64}.$$

In words, the above theorem shows that a function represented with polynomially many nodes at depth on the order of k^3 can require exponentially many nodes to reach even a fixed constant error at depth k. We remark that tighter depth-separation results (i.e., in the best case, showing an exponential gap between depth-$k+1$ and depth-k) remains open.

3 Learning neural networks: backpropagation

3.1 A 1000-foot view of machine learning

A supervised training data set is a collection

$$S = \{(x_i, y_i) : i = 1, \dots, n\},$$

where $x_i \in \mathcal{X}$ is an input from an input space $\mathcal{X}$, and $y_i \in \mathcal{Y}$ is its target from an output space $\mathcal{Y}$. We choose a parameterized neural network f_θ and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ for which $\ell(f_\theta(x_i), y_i)$ measures the error on example i. The empirical training objective is

$$\mathcal{L}(\theta) := \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i). \tag{9}$$

Stochastic gradient descent (SGD) approximately minimizes (9). At iteration t, choose a mini-batch $B_t \subseteq \{1, \dots, n\}$ and a learning rate $\eta_t > 0$, and update

$$\theta_{t+1} = \theta_t - \eta_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla_\theta \ell(f_{\theta_t}(x_i), y_i). \tag{10}$$

When B_t is sampled uniformly, the mini-batch gradient is an unbiased estimate of the full gradient under the usual sampling convention. Equation (10) leaves one computational question: how do we efficiently obtain the gradient with respect to every weight and bias in a deep composition?

3.2 Backpropagation

We first formalize the gradient-computation question using a *computational graph* (which may correspond to the computation executed by a neural network). Let $G = (V, E)$ be a directed acyclic graph whose scalar-valued nodes are ordered as $u^{(1)}, \dots, u^{(m)}$. The first m_0 nodes are inputs or parameters. For each $i > m_0$, let $\mathrm{Pa}(i) \subseteq \{1, \dots, i-1\}$ be the set of indices of the parents of node i. Each $u^{(i)}$ computes some function of the values computed by its parents; we let $\phi_i : \mathbb{R}^{|\mathrm{Pa}(i)|} \to \mathbb{R}$ be this function. It is straightforward to see that, e.g., an MLP may be expressed of this form (where each node in an earlier layer is ordered before each node in a later layer). The *forward pass* of such a computational graph evaluates

$$u^{(i)} = \phi_i\big((u^{(j)})_{j \in \mathrm{Pa}(i)}\big), \qquad i = m_0 + 1, \dots, m. \tag{11}$$

Assume the last node is the scalar loss $J := u^{(m)}$.

Recall that we want to compute the partial derivatives $\frac{\partial J}{\partial u^{(j)}}$, for nodes j corresponding to parameters. The chain rule, together with acyclicity, suggest a natural way of doing this; in particular, we have

$$\frac{\partial J}{\partial u^{(j)}} = \sum_{i:\ j \in \mathrm{Pa}(i)} \frac{\partial J}{\partial u^{(i)}} \cdot \frac{\partial u^{(i)}}{\partial u^{(j)}}. \tag{12}$$

Thus, we could try to proceed as follows: we can evaluate $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ directly since it corresponds to an explicit computation at one node (e.g., derivative of a ReLU or linear function). And then, for each i with $j \in \mathrm{Pa}(i)$, we can recursively find $\frac{\partial J}{\partial u^{(i)}}$; see the below algorithm.

Algorithm 1 Recursive derivative computation without dynamic programming

Require: The evaluated scalar computational graph from (11) and a node index $j \in \{1, \dots, m\}$
Ensure: The derivative $\partial J / \partial u^{(j)}$

function RecursiveDerivative(j)
  if $j = m$ **then**
    return 1
  end if
  $g \leftarrow 0$
  for all i such that $j \in \mathrm{Pa}(i)$ **do**
    $g \leftarrow g + \text{RecursiveDerivative(i)} \cdot \frac{\partial u^{(i)}}{\partial u^{(j)}}$
  end for
  return g
end function

Algorithm 1 suffers from a key drawback: it may recompute the derivative at the same downstream node once for every directed path leading to that node. For example, a layered graph with two nodes per layer and all four edges between each pair of consecutive layers has only $O(L)$ nodes and edges but $2^{\Theta(L)}$ directed paths. The recursion tree can therefore have exponential size, even though the computational graph itself has linear size.

Backpropagation is the dynamic-programming implementation of this same recursion, which takes time that is linear in the number of edges of the computational graph. The algorithm first evaluates and stores all node values in a forward pass, and then computes every adjoint in a single reverse pass (Algorithm 2).

Algorithm 2 Backpropagation on a scalar computational graph

Require: A DAG with nodes $u^{(1)}, \dots, u^{(m)}$ in topological order, local functions $(\phi_i)_{i=m_0+1}^{m}$, and scalar loss $J = u^{(m)}$
Ensure: The adjoints $\overline{u}^{(j)} = \partial J / \partial u^{(j)}$ for every node

Compute and store $u^{(m_0+1)}, \dots, u^{(m)}$ using (11)
Set $\overline{u}^{(j)} \leftarrow 0$ for $j = 1, \dots, m$
Set $\overline{u}^{(m)} \leftarrow 1$
for $i = m, m-1, \dots, m_0+1$ **do**
  for all $j \in \mathrm{Pa}(i)$ **do**
    $\overline{u}^{(j)} \leftarrow \overline{u}^{(j)} + \overline{u}^{(i)} \dfrac{\partial u^{(i)}}{\partial u^{(j)}}$
  end for
end for
return $\left(\overline{u}^{(1)}, \dots, \overline{u}^{(m)}\right)$

Correctness is immediate from Equation (12). Summarizing, in a scalar graph where each local partial derivative takes constant time, the backward pass takes $O(|E|)$ time, the same order as the forward pass.

For vector- or tensor-valued nodes, the same recursion replaces each scalar product in the inner loop of Algorithm 2 by a vector–Jacobian product. The principle is unchanged: cache the forward values, initialize the derivative of the loss with respect to itself to 1, and propagate derivatives backward through each local operation.

As a concrete example, consider a one-hidden-layer scalar-output network. Let $x \in \mathbb{R}^d$ be its input,

let the hidden width be $q \in \mathbb{N}$, and let $W \in \mathbb{R}^{q \times d}$, $b \in \mathbb{R}^q$, $v \in \mathbb{R}^q$, and $c \in \mathbb{R}$ be its parameters. Its forward pass is

$$z = Wx + b, \qquad h = \rho(z), \qquad s = v^\top h + c, \qquad J = \ell(s, y),$$

where ρ is applied coordinatewise. Define the scalar $\delta := \partial \ell(s, y)/\partial s$. For vectors of equal length, let $a \odot b$ denote their coordinatewise product, and let $r \in \{0, 1\}^q$ have coordinates $r_j = \mathbf{1}\{z_j > 0\}$. We take the ReLU derivative at $z_j = 0$ to be 0. Applying backpropagation in reverse order gives

$$\begin{aligned} \nabla_v J &= \delta h, & \frac{\partial J}{\partial c} &= \delta, \\ \nabla_z J &= (\delta v) \odot r, & \nabla_b J &= \nabla_z J, \\ \nabla_W J &= (\nabla_z J) x^\top. \end{aligned}$$

These are precisely the gradients inserted into the SGD update (10).

A Functional-analytic facts

This appendix records the two functional-analytic theorems invoked in the proof of Theorem 2.2.

Theorem A.1 (Hahn–Banach separation form)**.** *Let X be a real normed vector space, let $M \subseteq X$ be a closed linear subspace, and let $x_0 \in X \setminus M$. There is a continuous linear functional $\Lambda : X \to \mathbb{R}$ such that $\Lambda(m) = 0$ for every $m \in M$ and $\Lambda(x_0) = 1$.*

Theorem A.2 (Riesz representation theorem for $C(K)$)**.** *Let K be a compact Hausdorff space. For every continuous linear functional $\Lambda : C(K) \to \mathbb{R}$, there is a unique finite signed regular Borel measure μ on K such that*

$$\Lambda(f) = \int_K f \, d\mu \qquad \text{for every } f \in C(K).$$

References

[1] H. D. Block, B. W. Knight, Jr., and Frank Rosenblatt. Analysis of a four-layer series-coupled perceptron. II. *Reviews of Modern Physics*, 34(1):135–142, 1962. doi:10.1103/RevModPhys.34.135.

[2] Avrim Blum. The perceptron algorithm. Lecture 4, 15-859(B) Machine Learning Theory, Carnegie Mellon University, January 28, 2008. Course notes (PDF).

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. doi:10.1007/BF02551274.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Book website.

[5] Ingo Gühring, Mones Raslan, and Gitta Kutyniok. Expressivity of deep neural networks. arXiv:2007.04759, 2020. arXiv abstract page.

[6] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[7] Hrushikesh N. Mhaskar. Neural networks for optimal approximation of smooth and analytic functions. *Neural Computation*, 8(1):164–177, 1996. doi:10.1162/neco.1996.8.1.164.

[8] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi:10.1037/h0042519.

[9] Matus Telgarsky. Benefits of depth in neural networks. In *Proceedings of the 29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539, 2016. PMLR page.

[10] Haohan Wang and Bhiksha Raj. On the origin of deep learning. arXiv:1702.07800, 2017. arXiv abstract page.

AI Use. I prepared these lecture notes with the aid of AI assistance; I take full responsibility for all content in the notes.