Chapter 1

Introduction

1.1 Questions

A theory of neural networks should explain how neural networks work, in such a way that lets us make predictions about how well they work in what situations. Investigation of this question is often divided into three subquestions: expressivity, optimization, and generalization.

A neural network, like just about any other model in machine learning, is some function $f(\mathbf{x};\theta)$ where $\mathbf{x}$ is the input and θ contains the parameters to be learned. We want to train f on example inputs and outputs from some true function $f^*(\mathbf{x})$.

Expressivity asks whether the family of functions $f(\mathbf{x};\theta)$ for all θ includes f^*, or how close f can possibly get to f^* (in which case this could be called *approximation*, which has the advantage of rhyming with the other two). Let θ^* be the setting of θ that gets the closest.

Optimization asks whether the procedure we use for training f from some initial θ, on unlimited data, can reach θ^*, or whether it can fall short and reach some other parameter setting that is not as good.

Generalization asks whether training f on limited data can get as close to f^* as training f on unlimited data does, or whether it can overfit, approximating f^* well on the limited training data but poorly on test data.

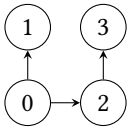
Exercise 1.1. What are your reasons for being interested in the theory of neural networks? (There aren't right or wrong answers here.)

1.2 Our focus: theory of computation

Our focus this semester (which might or might not be the focus of future offerings of this course) will be on what the *theory of computation* has to say about the questions posed above. The theory of computation studies various computational problems and how they can be solved in various models of computation (automata, Turing machines, families of Boolean circuits, logics).

For example: there are some problems that seem like they should be easy, but current large language models struggle with them, like multiplying two integers. Can we use theory to make predictions about whether we should expect language models to be able to solve such problems, or how language models could be improved in order to solve them?

The theory of computation traditionally studies computational problems as *formal languages*, or sets of finite strings over a finite alphabet. For example, multiplication problems can be represented as strings like `12*12=132`, where the problem is to decide whether the statement is true or not. Another example would be deciding whether a directed graph is strongly connected or not. A directed graph does not obviously have the form of a string, but can be represented (say) as an adjacency list in JSON format:




```
{0: [1,2], 1: [], 2: [3], 3: []}
```

Although this might seem somewhat unnatural, it actually fits rather well with current practice. Language models were originally developed for natural languages (whose utterances really are finite strings), and to use language models on complex structures, we often simply serialize them, as above, and hope that the model can "deserialize" them.

One consequence of our focus on strings is that, although we will start with feed-forward neural networks, which operate on fixed-size objects, we will proceed rather quickly to neural networks that operate on strings of variable, unbounded length. If we were to only consider strings of length up to (say) 1000, we would not be able to separate computational problems, or models of computation like neural networks, in any interesting way. While pragmatists may say that astronomically long strings are unrealistic, I'd say that this again fits rather well with current practice. The latest language models (at the time of writing) have a maximum context length that is over a million tokens, and growing fast.

1.3 Overview

(This offering of) the course will have four units. After a warm-up on perceptrons considering the three questions of expressivity, optimization, and generalization, we will study the expressivity of three major kinds of neural networks:

1. **Feedforward neural networks** (FFNNs) are the most basic architecture, and a key component of the other two architectures we will look at.
2. **Recurrent neural networks** (RNNs) are the classic architecture for modeling sequential data. They dominated natural language processing for several years, and could be making a comeback with variants like RWKV and state-space models.
3. **Transformers** currently dominate natural language processing and other application areas of neural networks as well.

Then the last unit will be on

4. **Optimization and generalization.** We'll give an overview of the key results in optimization and generalization for neural networks. Most research in this area focuses on FFNNs, and so will we, but we will try to review some recent results on RNNs and transformers as well.

1.4 Preliminaries

We write $\mathbb{N}$ for the set of natural numbers including 0, and $\mathbb{N}_{>0}$ for the natural numbers excluding 0. We write $[n]$ for the set $\{0, \dots, n-1\}$ (note: not $\{1, \dots, n\}$).

We write $\log x$ for the natural logarithm of x and $\log_2 x$ for the base-2 logarithm of x. In the expression $O(\log n)$, the base does not matter, so we omit it. We write either e^x or $\exp x$ for the exponential function and sometimes we write $\exp_2 x$ for 2^x. We write $O(\mathrm{poly}(n)) = \bigcup_{k \geq 0} O(n^k)$.

1.4.1 Linear algebra

Variables that stand for vectors are lowercase boldface letters: $\mathbf{a}, \mathbf{b}, \dots$. We write $\mathbf{0}$ for the zero vector. Variables that stand for matrices are uppercase boldface letters: $\mathbf{A}, \mathbf{B}, \dots$.

If $\mathbf{x} \in \mathbb{R}^d$, we will normally write the components of $\mathbf{x}$ as $x_0, \dots, x_{d-1}$ (note: 0-based indexing). But sometimes this is not convenient, so we also use a more code-like notation where the components of $\mathbf{x}$ are $\mathbf{x}[0], \dots, \mathbf{x}[d-1]$. If we write $\mathbf{x}_0, \mathbf{x}_1$, these are names of two different vectors, and similarly for $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}$. So,

subscript $_i$ and superscript $^{(i)}$ mean "the i-th thing," but bracketed $[i]$ means "the i-th element of."

For $i \in [d]$, we write $\mathbf{e}_i$ for the i-th unit vector of the standard basis of $\mathbb{R}^d$, that is, the vector with a 1 in the i-th component and 0 everywhere else.

1.4.2 Strings

If A is any set, we write A^* for the set of finite sequences of elements of A. If Σ is a finite alphabet (set of symbols), then Σ^* is called the set of strings over Σ. We also often deal with sequences of vectors; we write (nonstandardly) $(\mathbb{R}^d)^*$ for the set of finite sequences of vectors in $\mathbb{R}^d$.

If $\mathbf{a} \in A^*$, we write $|\mathbf{a}|$ for the length of $\mathbf{a}$ and a_i or $\mathbf{a}[i]$ for the i-th element of $\mathbf{a}$. As with vectors, we number the elements starting from 0, not 1. If $\mathbf{a}, \mathbf{b} \in A^*$, we write $\mathbf{ab}$ or sometimes $\mathbf{a} \cdot \mathbf{b}$ for the concatenation of $\mathbf{a}$ and $\mathbf{b}$.

A function from A^* to B^* is *length-preserving* if for all $\mathbf{a} \in A^*$, we have $|\mathbf{a}| = |f(\mathbf{a})|$. In this case, we write $f\colon A^* \xrightarrow{\text{lp}} B^*$.

1.4.3 Miscellaneous

Iverson bracket For any true or false statement ϕ, we write

$$\mathbb{I}[\phi] = \begin{cases} 1 & \text{if } \phi \text{ is true} \\ 0 & \text{if } \phi \text{ is false.} \end{cases} \tag{1.1}$$

Exercise 1.2. Did you catch these idiosyncratic notations?

(a) If $\mathbf{x}$ is a vector, what is its first component: $\mathbf{x}_0$, $\mathbf{x}_1$, x_0, x_1, $\mathbf{x}[0]$, $\mathbf{x}[1]$, $x[0]$, $x[1]$? There may be more than one correct answer.

(b) If $\Sigma = \{\mathsf{a}, \mathsf{b}\}$ and $f\colon \Sigma^* \xrightarrow{\text{lp}} \Sigma^*$, then what is $|f(\mathsf{aaa})|$?

(c) Let

$$\mathbf{X} \in (\mathbb{R}^d)^*$$
$$\mathbf{X} = \begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}4\\5\\6\end{bmatrix}\begin{bmatrix}7\\8\\9\end{bmatrix}.$$

What is $\mathbf{X}[1][2]$? What is $\mathbf{X}[2][1]$?