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 rhymes 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.

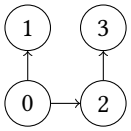
Generalization asks whether training f on limited data can get as close to f^* as training f on unlimited data does, or whether it can overfit, approximating f^* well on the limited training data but poorly on test data.

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 what improvements to language models could be made 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, a graph can be represented as an adjacency list in JSON format.




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

Although this might not always be the most natural representation of complex data structures, this 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.

One consequence of our focus on strings is that we will start with feedforward neural networks, which operate on fixed-size objects, but proceed rather quickly to neural networks that operate on strings. And we will focus on results that are true for strings of *unbounded length*. 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

The course will have three major units:

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 sev-

eral 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.

Most research on neural network theory focuses on FFNNs, and I will try to give a well-rounded introduction to this research. But research on the theory of RNNs and transformers has focused on expressivity much more than optimization or generalization. Consequently, as we move on to RNNs and transformers, our focus, too, will be almost exclusively on expressivity.