Lecture 20: Neural Networks – Optimization

Instructor: Aditya Bhaskara *Scribe: Aashish Gottipati*

CS 5966/6966: Theory of Machine Learning

March 29th, 2022

Abstract

In this lecture, we explore various aspects of neural network optimization. Specifically, we touch on gradient descent, overparameterization, and feature learning.

1 Neural Networks (DNN)

1.1 definition. A layered "circuit" that takes a vector of input features x, produces output $y = F_r \circ F_{r-1} \circ ... \circ F_1(x)$, where each F_i is a function of the form $F_i(z) = \sigma(Az + b)$, for some activation function $\sigma()$ (that acts coordinate-wise).

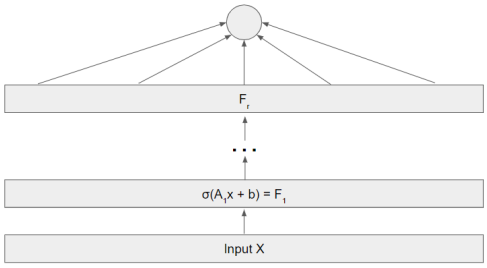

Figure 1: Example Neural Network

Here are some examples of commonly used activation functions:

- Threshold
- Sigmoid (continuous approximation): $\frac{1}{1+e^{-x}}$
- ReLU and Tanh

2 Learning Neural Networks

In previous lectures, we mentioned that neural networks can represent and approximate any function by the universal approximation theorem. Consequently, we can formulate the supervised learning **problem** as follows: given data (x_1, y_1) , (x_2, y_2) , ..., (x_m, y_m), where $x_i \in \mathbb{R}^n$ and y_i denotes the label, from some distribution D, find h (with given "architecture") that minimizes the risk. In the context of neural network learning, the Empirical Risk Minimization (ERM) problem is usually referred to as neural network training.

See (Barron, Cybenko) for more information on the universal approximation theorem

Choosing the network architecture is difficult and depends on the type of problem (inductive bias). Typically, we rely on heuristics for choosing our architecture, e.g., convolutional neural networks (CNN) for image data and transformers for natural language processing. However, even with these loose guidelines, many other architecture decisions must be made (depth vs. width trade offs). As a result, there is no "good" general purpose technique for selecting the architecture.

2.1 THEOREM. *Given an architecture, it is NP-hard to learn weights, even if classification error is* 0 *and we just have* 3 *internal nodes.*

See the textbook for more details on Theorem 2.1

In essence, Theorem 2.1 states that even if we are given an architecture, finding the optimal weights is NP-hard. However, clearly this is not reflected in practice and is a worst case result. Naturally the question becomes, can we obtain more "positive" theoretical results (not just worst case)?

It turns out that this problem is fairly difficult and is still an open problem. However, if the width is large, it has been shown that you can train efficiently, assuming a depth of 2 and random inputs (see Theorem 2.2).

2.2 THEOREM. *Given* $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$ *where* $x_i \in \mathbb{R}^n$ *and* $y \in \{0, 1\}$, *decide if there exists a network of the structure in Figure 2 such that* $h(x_i) = y_i \ \forall_i$

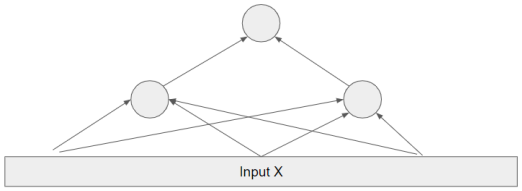

Figure 2: Shallow Network

However, theorem 2.2 suffers from the fact that it relies on "worst" case inputs (i.e., corrupted), while also constraining the network size to be "too small". Follow up works have tried to remedy these weaknesses by exploring scenarios where inputs are drawn from $\mathcal{N}(0,1)^n$. For example, consider an unknown/hidden network where its input $x \sim \mathcal{N}(0,1)^n$. Then, by examining, the input and output behaviors of the unknown network ("correlations" between y and x), we may be able to uncover the architecture of the hidden network (i.e., find the network that generates y from x). On the other hand, other works have explored relaxing the problem to allow slightly larger architectures.

In practice, the gradient descent (GD) algorithm is used to train neural networks, since computing gradients is not too difficult and is simply an application of the chain rule. Specifically, this algorithm is known as "Back propagation" and runs in linear time.

See (Rumelhart, Hinton, Williams) for more information.

3 Is Gradient Descent Good?

In terms of runtime, naive GD takes time roughly $N \times |w|$ per iteration, where $|w|$ is the number of weights, which isn't very good. This can be seen from the

fact that our loss for a single interation is given by

$$L(x, w) = \sum_{i=1}^{N} l(x_i, w)$$

for data $(x_1, y_1), ..., (x_n, y_n)$. As a result, in practice, a form of stochastic GD is utilized. We divide N into "batches" and compute the gradient only using a batch, which greatly reduces the runtime per iteration.

In terms of optimality, given that we can run GD, there are two questions we need to answer. First, given data $(x_1, y_1), (x_2, y_2), ...$, does running GD for N iterations result in training error $\leq OPT + f(N)$ for some decreasing function? In this case, the answer is **no** because this function is not convex. Second, assuming the network architecture allows for zero error, does GD converge to zero error? The answer is **no** (see NP-hardness result in Theorem 2.1).

Consequently, much work was put into researching alternative moments-based variants of GD; however, these techniques tend to only work well for shallow networks. In practice, we still do not have a good alternative to gradient descent yet.

See (Anankermas et al.) for more information on shallow network analysis. (Chen, Klivans, Meka 2020) show that in time exp(# internal nodes, depth, other parameters) moment-based variant can learn what GD can't (fixed parameter tractability).

4 Overparameterization

Given that when trying to derive bounds for GD on non-convex functions, things fail badly, many people turned to the following question: can we show that GD is good in any reasonable generality?

In some cases, people have been able to show that for over parameterized networks (width of network $\geq$ some # inputs), then all local optimiums are close to global optimums (see Theorem 4.1).

4.1 theorem. *A width of $\sim n^3$ network with any number of layers trained via GD from random initialization achieves zero training error with high probability.*

See (Jacot, Gabriel, Hongler 18) and (Arora, et al. 2019) for more information on Theorem 4.1

The key idea is that the parameters do not change much during training if the width is very large. Intuitively, when the width of network is so large, then every data point can be memorized. Nonetheless, the main takeaway is that this is possible with GD, even though it appears to be just memorization.

This relates to the idea of neural tangent kernels. Informally, the idea of neural tangent kernels is as follows: GD-based training works like a kernel method with an appropriately defined kernel (at least with infinite width).

5 Feature Learning

When we started with neural networks, we wanted to start with some input x, derive some basic features from x, derive some complex features from our basic features, then finally derive some classification features for our task of classification (see Figure. 3).

In essence, our neural network is performing linear classification in "feature space." We can view our neural network as embedding x in some feature space, which leads to the problem of feature learning.

Our feature learning problem is as follows, given some inputs, can we "extract useful features"? The answer is **yes**. Some examples of feature extraction techniques are as follows:

- Contrastive learning

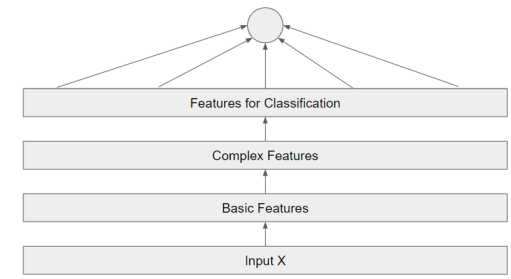

Figure 3: General Neural Network Feature Extraction Flow

- Discriminative learning (predicting classes that are orthogonal to each other)
- Manifold learning (every point defined by $k << N$) parameters
- Spare coding (autoencoding)

Based on the embedding view of neural networks, the goal is to find a feature embedding mapping of data points $x \rightarrow f(x)$, where x is our raw data and $f(x)$ is our feature embedding, that satisfies certain properties. Our embeddings should have the following properties:

1. $f(x)$ should be useful to discern between classes
2. Coordinates of $f(x)$ should be ”independent“ of one another
3. $f(x)$ should be “as informative” about x as possible

As a result, knowing $f(x)$ we should be able to “recover” x to a certain extent.