Lecture #: Gradient Descent Dynamics, Neural Tangent Kernel

Instructor: Aditya Bhaskara *Scribe: Ian Argyle*

CS 5966/6966: Theory of Machine Learning

March 31st, 2022

Abstract

In this lecture, we show that gradient descent is equivalent to kernel regression with a time varying "tangent" kernel. We also discuss why a sufficiently over-parametrized model with random initialization can achieve zero error.

1 Introduction

To recap, recall that given some data $(x_1, y_1), (x_2, y_2), \ldots$ from some distribution D, we want to find h that minimizes the risk. This is hard to solve in exactly, so in practice we use gradient descent.

Our goal is to reason about the dynamics of gradient descent, and the solution it provides. We can start by asking a question: is gradient descent easier to analyze when the network is heavily overparametrized?

2 Over-Parametrization

Recall that we define a model as being overparametrized if the number of parameters in the model is much larger than the size of the training data.

This brings us to the following theorem:

2.1 THEOREM. *A width $\approx N^3$ network (any number of layers) trained via GD from random initialization achieves zero training error. Moreover, the final solution is equivalent to solving a "Kernel regression" problem with a specific kernel.*

We will examine this is more depth, but first let's review what kernel regression is.

3 Kernel Regression

Recall that in regression problems, we are given the value of a function at a finite number of points, and we want to try to interpolate the rest of the space.

Kernel regression is based on a similarity metric between two points, this is called the kernel $K(x, y)$. Using this method, if we want to interpolate the function value at a specific point x, we can use an equation in the following form:

$$\hat{f}(x) = \sum_{i=1}^{N} \alpha_i K(x, x_i)$$

So, the goal with kernel regression is to find the best α's given some kernel (such as Gaussian). Mathematically, we want to solve for $\alpha_1, \alpha_2, ...\alpha_N$ where $\hat{f}(x) = y$.

To achieve this, consider an NxN matrix K where $K_{i,j}$ is defined as $K(x_i, x_j)$, if we put the y's and α's into vectors, we get the following:

$$\begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K(x_N, x_1) & \dots & K(x_N, x_N) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Notice that this has a closed form solution: $\alpha = K^{-1}y$. However, in practice it's better to use gradient descent to minimize $g(\alpha) = \|K\alpha - y\|$ (we did this in the last homework). Specifically, this means initialize $\alpha^{(0)}$, then

$$\alpha^{(t+1)} = \alpha^{(t)} - 2\eta K(K\alpha - y)$$

for some learning rate η since $\nabla g^{(a)} = 2K(K\alpha - y)$.

How does this relate to neural networks? First, lets analyze how $\hat{f}$ changes over time (as you do steps of gradient descent):

$$\begin{aligned} \hat{f}(\alpha^{(t+1)}, x) - \hat{f}(\alpha^{(t)}, x) &= \sum_{i=1}^{N} \alpha_i^{(t+1)} K(x, x_i) - \sum_{i=1}^{N} \alpha_i^{(t)} K(x, x_i) \\ &= -\eta \langle \nabla \hat{f}(\alpha^{(t)}, x), \nabla g(\alpha^{(t)}) \rangle \\ &= -\eta \langle \nabla \hat{f}(\alpha^{(t)}, x), \sum_{i=1}^{N} (\hat{f} - y)_i \nabla \hat{f}(\alpha^{(t)}, x_i) \rangle \\ &= -\eta \sum_{i=1}^{N} (\hat{f} - y)_i K(x, x_i) \end{aligned}$$

4 Gradient Descent Analysis

Now, let's analyze how some model $f(w, x)$ changes as we do steps of gradient descent. That is:

$$\begin{aligned} f(w^{(t+1)}, x) - f(w^{(t)}, x) &= f(w^{(t)} - \eta \nabla L(w^{(t)}), x) - f(w^{(t)}, x) \\ &= f(w^{(t)} - \varepsilon, x) - f(w^{(t)}, x) \text{ setting } \eta \nabla L(w^{(t)}) = \varepsilon \end{aligned}$$

Now, we know that $f(w + \varepsilon, x) - f(w, x) \approx \langle \nabla f(w, x), \varepsilon \rangle$ so, we have:

$$f(w^{(t)} - \varepsilon, x) - f(w^{(t)}, x) = -\eta \langle \nabla f(w^{(t)}, x), \nabla L(w^{(t)}) \rangle$$

Notice that $\nabla_w L(w) = \sum_{i=1}^N (f(w, x_i) - y_i)\nabla_w f(w, x_i)$. Setting $f(w^{(t)}, x_i) = u_i^{(t)}$ leaves us with:

$$\begin{aligned} f(w^{(t)} - \varepsilon, x) - f(w^{(t)}, x) &= -\eta \langle \nabla f(w^{(t)}, x), \sum_{i=1}^N (u_i^{(t)} - y_i)\nabla_w f(w^{(t)}, x_i) \rangle \\ &= -\eta \sum_{i=1}^N (u_i^{(t)} - y_i) \langle \nabla f(w^{(t)}, x), \nabla f(w^{(t)}, x_i) \rangle \end{aligned}$$

Defining $\langle \nabla f(w^{(t)}, x), \nabla f(w^{(t)}, x_i) \rangle = H^{(t)}(x, x_i)$ leaves us with:

$$f(w^{(t+1)}, x) - f(w^{(t)}, x) = -\eta \sum_{i=1}^N (u_i^{(t)} - y_i) H^{(t)}(x, x_i)$$

Now, we can see that this is equivalent to kernel regression with a time-varying "tangent" kernel ($H^{(t)}$).

5 Other Findings

While we didn't show this in class, the authors of the paper mentioned above (Jacot et al.) also showed that for very wide, randomly initialized neural networks, the kernel remains "nearly fixed" with time.

Also, for a "large" amount of time, $w^{(t)}$ doesn't change much. Intuitively this makes sense, if there are sufficiently many parameters, w doesn't need to change very much to achieve zero error. However, in this case, w's won't correspond to useful features.