Unnamed Thesis

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

April Kopec

May 2024

Introduction

Neural networks have become a fundamental tool in the field of machine learning. But despite the fact that the lower-level implementation of a neural network is relatively simple, the behavior of neural networks has proven very difficult to analyze at a higher level. It has been estimated that GPT-4, a state-of-the-art large language model, has over one trillion parameters—though OpenAI has not officially disclosed the precise number. The neural networks used in the modern day contain vast amounts of data, and use that data in an opaque enough manner that the field has struggled to understand much about their behavior.

One possible avenue for understanding neural networks better is to analyze the relationship between the data they are trained on and their end behavior. What predictions can we make about how the final model would have been different if it had been trained on different data? *Influence functions* are one approach through which we can try to answer this question.

At a high level, the idea behind an influence function is to calculate how much a very small change in the weighting of one training data point influences the resulting model. To discuss influence functions in more depth, we will first introduce some fundamental ideas in machine learning. These include parametric models, loss functions, and gradient descent.

0.1 Parametric Models

Suppose we have some data and want to produce a computer algorithm which can answer questions about our data. For instance, we might try to predict test performance based on how long someone spent studying, or someone's wealth based on their age. More usefully, a credit bureau might calculate your credit score (meant to measure likelihood of defaulting on loans), or an LLM might try to predict the next batch of characters based on the previous characters. An algorithm designed for this sort of task can be referred to as a "predictive model"—by using some input data, the model makes a prediction about a related variable.

But how does one decide on the best model to use when dealing with some prediction task? This can be done using *training data*, or a large collection of examples of input data the model might receive and the corresponding desired outputs. A training data set can be thought of as a series $\mathcal{D} = \{(x_i, y_i)\}$ consisting of pairs of input data $x_i \in \mathcal{X}$ and outputs $y_i \in \mathcal{Y}$.

However, merely choosing whichever function $M : \mathcal{X} \to \mathcal{Y}$ is best at predicting the training data would not work well. A model which only memorizes a lookup table of which y_i corresponds to which x_i in the training data would perform perfectly during training, but would be nearly useless in practice. Our model should generalize to inputs outside of the training data, rather than only being effective on the exact examples it has already seen. (When a model fails to achieve this, it is known as *over-fitting.*)

Instead, we can choose a function from a *parameterized family of models.* The hope is that we can specify a family of models which use parameters $\theta_1, \theta_2, \cdots$ in a fixed manner to produce predictions. By confining the model to a particular form, we hope that it will have to learn generalizable properties about the training data, rather than just memorizing the specific training examples.

Definition 0.1. A **model** is a function $M : \mathcal{X} \to \mathcal{Y}$. The inputs $x \in \mathcal{X}$ can be thought of as collections of data about some question, and the outputs $M(x) \in \mathcal{Y}$ can be thought of as the model's answer. A **parameterized family of models** consists of a set $\mathcal{M}$ of models, a *parameter space* Θ, and a function which specifies a model $M_\theta \in \mathcal{M}$ given parameters $\theta \in \Theta$. If we have specified a family, we can use $M_\theta(x)$ to denote the output of the model specified by parameters θ when receiving the input x. If a model is a member of a parameterized family, it can be called a **parameterized model**.[4]

It is often reasonable to assume that $\mathcal{X}, \mathcal{Y}$, and Θ are finite-dimensional vector spaces over $\mathbb{R}$—later, we will be applying vector calculus. (Though also note that sometimes it makes more sense to only consider a subset of the vector space. For example, if $y \in \mathcal{Y}$ is meant to represent a probability distribution over a discrete set, the components of y ought to sum to one.)

This definition is fairly abstract, so it may be helpful to consider a few concrete examples.

1. In simple linear regression, the model aims to predict some (real) number y using a linear function of another number x. We do this using a models of the form $y = \beta_0 + \beta_1 x$. Here, β_0 can be interpreted as the *intercept*, and β_1 can be interpreted as the *slope*. In this case, $\mathcal{X}$ is $\mathbb{R}$, $\mathcal{Y}$ is $\mathbb{R}$, and $\Theta = \mathbb{R}^2$ contains vectors of the form $\theta = (\beta_0, \beta_1)$.

2. Linear regression can be generalized to multiple linear regression, where $\mathcal{X} = \mathbb{R}^n$. In this case, we have $y = \beta_0 + \beta \cdot x$, and $\Theta = \mathbb{R} \times \mathbb{R}^n$. Going further in the same direction, it can be generalized to to multivariate linear regression, where $\mathcal{Y} = \mathbb{R}^m$, $y = \beta_0 + Bx$ for a matrix B, and $\Theta = \mathbb{R}^m \times \mathcal{M}_{n\times m}$. A great many other generalizations are possible.

3. In logistic regression, we try to estimate the probability of some event Y based on a real number x. We use models of the form $\mathrm{P}[Y] = \dfrac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$. In this case, $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = (0,1)$, and $\Theta = \mathbb{R}^2$. In logistic regression, it is still possible to think of $\mathcal{Y}$ as a vector space; you can have the model instead output $\operatorname{logit}(\mathrm{P}[Y]) = \ln \dfrac{\mathrm{P}[Y]}{1-\mathrm{P}[Y]} = \beta_0 + \beta_1 x$. Much like linear regression, logistic regression has several possible generalizations.

4. Neural networks are too complicated to express using a simple equation, but the set of neural networks (of a fixed size) is a parametric family. For a neural network, $\mathcal{X}$ is a vector space where each component of x is the activation of a neuron in the input layer, $\mathcal{Y}$ is the same for the output layer, and Θ contains large vectors whose components each represent one weighting of a connection between neurons. If you hear someone refer to the "parameter count" of a neural network, they are referring to $\dim \Theta$ (or equivalently to the number of entries of a vector $\theta \in \Theta$.)

From the breadth of the examples, it is clear that this is a very general framework.

0.2 Training Models

0.2.1 Loss Functions

Given a parameterized family of models, we now need a method of selecting the best model. A first guess at an approach might be to choose the model that makes the most correct predictions when applied to the training data. However, typically we also care about answers that are *approximately* correct, not only about answers that are exactly correct. This means that to choose the best model, we need some method of "scoring" them which accounts for not just whether an answer is correct, but also how close it is to being correct. In machine learning, such a scoring method is referred to as a *loss function.*

Definition 0.2. Let $\mathfrak{D} = \mathcal{P}(\mathcal{X} \times \mathcal{Y})$ be the set of possible databases $\mathcal{D}$ of pairs $z_i = (x_i, y_i)$. A **loss function** is a function $\mathcal{L} : (\mathfrak{D} \times \Theta) \to \mathbb{R}$ where $\mathcal{L}_{\mathcal{D}}(\theta)$ represents how *poorly* the model M_θ performs on the database $\mathcal{D}$ when it tries to predict the outcomes y_i given x_i. (When working with a fixed data set $\mathcal{D}$, we can leave it implicit and simply refer to $\mathcal{L}(\theta)$.) A **pointwise loss function** is a function $\ell : ((\mathcal{X} \times \mathcal{Y}) \times \Theta) \to \mathbb{R}$ such that $\mathcal{L}$ is equal to the average of ℓ across $\mathcal{D}$.

It is often, though not necessarily, the case that the loss function is simply a measure of the quality of the model's predictions. For example, consider linear regression. Here, $\ell_{z_i}(\theta) = (M_\theta(x_i) - y_i)^2$, the squared error. In cases like this, $\ell_{z_i}(\theta)$ can be expressed as $d(M_\theta(x_i), y_i)$ where d is some measure of prediction error.

Conversely, consider *ridge regression.* Ridge regression is a modified form of linear regression which also tries to choose *small* coefficients (or parameters), rather than just choosing whichever coefficients best predict the data. In the simplest case, ridge regression uses a loss function like

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (M_\theta(x_i) - y_i)^2 + \lambda \sum_j \theta_j^2,$$

where λ is a factor which determines how strongly to penalize a larger coefficient. This loss function has one term for the mean squared error, and a second term which incentivizes the model to choose smaller parameters.

Equipped with the concept of a loss function, we can now express mathematically what the best parameters are for a given parametric family and a given training data set.

Definition 0.3. Let θ^* be a tuple of parameters for a model M in some parametric family with a given loss function $\mathcal{L}$ and training data set $\mathcal{D}$. The parameters θ^* are **globally optimal** if

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{L}(\theta)$$

—that is, if there is no $\theta' \in \Theta$ such that $\mathcal{L}(\theta') < \mathcal{L}(\theta^*)$. If Θ has a topology, we say the parameters θ^* are **locally optimal** if there is a neighborhood of θ^* (imagine a region in Θ containing θ^* fully inside itself, rather than on the boundary) such that there are no better parameters θ' within that neighborhood.

Informally, suppose we imagine parameters as being a horizontal position, and the loss for the corresponding model as being the elevation of the ground at that position. Then θ^* is a local optimum if it is the lowest point in some particular valley, and it is the global optimum if is in the deepest trench in the entirety of Θ.

0.2.2 Gradient Descent

Given a parametric family, training data, and a loss function, how can you actually evaluate (or approximate) $\arg\min_\theta \mathcal{L}(\theta)$ and select the optimal parameters? One common and effective technique is known as gradient descent. At a high level, the idea is to repeatedly update your parameters by a small step in whichever direction decreases loss the most.

Definition 0.4. Consider $\mathcal{L}$ as a scalar field over $\Theta = \mathbb{R}^p$ (that is, a function from $\Theta \to \mathbb{R}$). Then the **gradient** of $\mathcal{L}$ with respect to θ is given by

$$\nabla_\theta \mathcal{L} = \begin{bmatrix} \dfrac{\partial}{\partial \theta_1} \\ \vdots \\ \dfrac{\partial}{\partial \theta_p} \end{bmatrix} \mathcal{L} = \begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial \theta_1} \\ \vdots \\ \dfrac{\partial \mathcal{L}}{\partial \theta_p} \end{bmatrix}$$

where $\dfrac{\partial \mathcal{L}}{\partial \theta_i}$ is the partial derivative of $\mathcal{L}$ with respect to the i-th component of θ. Because each partial derivative is a scalar field over Θ, the gradient is a vector field.

The key property of the gradient is that, for some fixed θ^*, the gradient $\nabla_\theta \mathcal{L}(\theta^*)$ is a vector in Θ which points in the direction where a movement of θ^* *increases* loss the quickest. Since the goal is to *minimize* cost, we will instead be interested in the opposite direction, which is given by $-\nabla_\theta \mathcal{L}(\theta^*)$.

The process of gradient descent is described by the following pseudocode:

Algorithm 1: Gradient Descent

```
1 for $i$ from 1 to $\dim \Theta$ do
2     Approximate $\frac{\partial \mathcal{L}}{\partial \theta_i}(\theta^*)$ by $\frac{\mathcal{L}(\theta^* + \frac{\varepsilon}{2} e_i) - \mathcal{L}(\theta^* - \frac{\varepsilon}{2} e_i)}{\varepsilon}$
3 Calculate $\nabla_\theta \mathcal{L}(\theta^*)$ using partial derivatives
4 Set $\theta^*$ to $\theta^* - \nabla_\theta \mathcal{L}(\theta^*)\delta$
5 Repeat
```

where e_i is the basis vector in the θ_i direction, and ε and δ are small positive reals. The variable δ is sometimes known as the *learning rate*.

This algorithm can be altered in many ways to make it more efficient—for example, lowering the learning rate δ over time as the model starts needing smaller refinements. But the high-level concept is always simply to follow the (negative) gradient down and hopefully approach a minimum.

0.3 Influence Functions

Suppose we want to consider how altering the training database $\mathcal{D}$ impacts the resulting model—in particular, we will consider *local* alterations, rather than large ones. For this, it will be helpful to define the concept of a *weighted* data set.

Definition 0.5. A **weighted data set** $\mathcal{D}$ is a sequence $\{(z_i, w_i)\}$ of data points z_i and associated weightings w_i. Conceptually, sums in an unweighted data set correspond to weighted sums, averages to weighted averages, etc:

$$\begin{array}{ccc} \sum f(z_i) & \textbf{corresponds to} & \sum w_i f(z_i) \\ \dfrac{\sum f(z_i)}{n} & \textbf{corresponds to} & \dfrac{\sum w_i f(z_i)}{\sum w_i} \end{array}$$

An unweighted data set can be thought of as a weighted data set where each w_i is equal to 1.

It is often useful to consider $\mathcal{D}$ as though it assigns a weighting to every possible data point $(x, y) \in (\mathcal{X} \times \mathcal{Y})$. In these cases, we say that the database assigns a weighting of $w = 0$ to each point not in its sequence; including a point in the sequence but assigning it a weight of zero is equivalent to excluding it.

Now let us fix a particular data point $z_m = (x_m, y_m)$, which will have some weighting w_m. Suppose we change the weighting by some increment ε. If z_m was already in the unweighted data set, then setting $\varepsilon = -1$ would be equivalent to removing it; if it was not, then setting $\varepsilon = 1$ would be equivalent to adding it.

For a fixed choice of training data set and z_m, we will consider how the final model changes as a result of changing ε.

Definition 0.6. Let $\mathcal{D}_\varepsilon$ denote the weighted data set where ε has been added to w_m. Then the **response** of θ to z_m is given by

$$R_m(\varepsilon) = \operatorname*{arg\,min}_{\theta \in \Theta} \mathcal{L}_{\mathcal{D}_\varepsilon}(\theta) = \operatorname*{arg\,min}_{\theta \in \Theta} \frac{\sum_{i=1}^n w_i \ell_{z_i}(\theta)}{\sum_{i=1}^n w_i} \tag{1}$$

The response of a function f to z_m is given by $f \circ R_m$. In particular, we might consider the response of loss $\mathcal{L}_\mathcal{D}(R_m(\varepsilon))$ or the response of predictions $M_{R_m(\varepsilon)}$.

The response function can be thought of as a curve through Θ parameterized by ε; the response of loss can be easily visualized as a two dimensional graph where the x-axis represents ε and the y-axis represents $\mathcal{L}$.

Definition 0.7. The **influence** of z_m on θ is the derivative of the response function, evaluated at $\varepsilon = 0$.

$$\mathcal{I}(z_m) = \left.\frac{\partial \theta}{\partial \varepsilon}\right|_{\varepsilon=0} \tag{2}$$

The influence of z_m on a differentiable function f of θ is the derivative of the response of f to z_m evaluated at $\varepsilon = 0$. Using the chain rule, we can calculate:

$$\mathcal{I}_f(z_m) = \left.\frac{\partial (f \circ R_{z_m})}{\partial \varepsilon}\right|_{\varepsilon=0} = \left.\frac{\partial f}{\partial \theta} \frac{\partial \theta}{\partial \varepsilon}\right|_{\varepsilon=0} = \frac{\partial f}{\partial \theta} \mathcal{I}(z_m) \tag{3}$$

where $\frac{\partial f}{\partial \theta}$ is evaluated at $\theta = R_{z_m}(0)$ (which is simply the value of θ before any reweighting). Note that if f is a scalar function of θ, then $\frac{\partial f}{\partial \theta}$ is the same as the gradient $\nabla_\theta f$; if it is a vector function, then it is a Jacobian matrix.[1][2][3]

If the response function is thought of as a curve through Θ, then the influence function would be the tangent vector to that curve at $\varepsilon = 0$; the influence function tells you the direction the curve is moving in at that point. If the curve is sufficiently straight, then the influence can be used to approximate the response function:

$$R_m(\varepsilon) \approx R_m(0) + \mathcal{I}(z_m)\varepsilon + O(\varepsilon^2).$$

(This is simply a Taylor approximation.)

The influence of z_m on pointwise loss $\ell_{(x_i,y_i)}$ or on the prediction $M_\bullet(x_i)$ for some particular data point can be thought of as measures of how much the training data point z_m contributed to the model's prediction for that particular data.

0.3.1 Example

To help make this more concrete, let us look at a very basic example: simple linear regression. Let the input space $\mathcal{X}$ and output space $\mathcal{Y}$ both be $\mathbb{R}$. Linear regression models are of the form $y = mx + b$, so $\Theta = \mathbb{R}^2$ and $\theta = (m, b)$. Our data set $\mathcal{D}$ will consist of some data points (x_i, y_i), and our loss function will be given by the squared error $\ell_{(x_i,y_i)}(m, b) = ((mx_i + b) - y_i)^2$.

Consider the data set $\mathcal{D} = \{(0, 0), (1, 1.5), (2, 2)\}$. The model we get with weighting each point equally is $y = 1x + 0.167$. We find that the influence of $(0, 0)$ on θ is $\mathcal{I}(0, 0) \approx (0.083, -0.13)$, that $\mathcal{I}(1, 1.5) \approx (0.00093, 0.11)$, and that $\mathcal{I}(2, 2) \approx (-0.083, 0.028)$.

This confirms what we expect: if you imagine adjusting the model slightly in the direction of one of these influences, you will find that the model predicts the data point in question slightly better at the cost of predicting the other two points slightly worse.

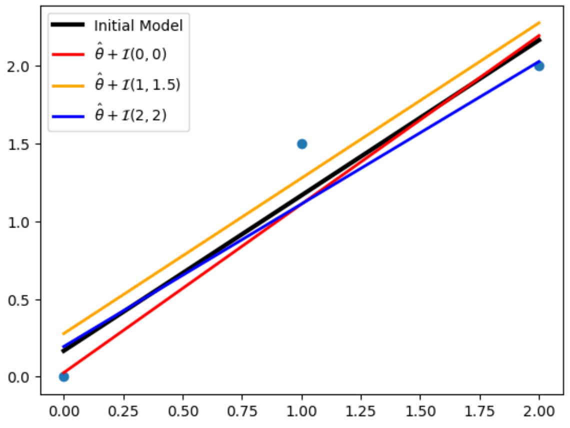

Figure 1: The three data points, along with four different models. The model in black is the model with equal weights. The other three models move the optimal model's parameters by one unit in the direction of $\mathcal{I}(z)$.

We can also consider the influence on cost. We could calculate the influence of cost for any hypothetical data point anywhere in the xy plane, but for now we will only use the training data points as our examples.

	$\ell_{(0,0)}$	$\ell_{(1,1.5)}$	$\ell_{(2,2)}$
$(0,0)$	-0.046	0.037	0.009
$(1,1.5)$	0.037	-0.074	0.037
$(2,2)$	0.009	0.037	-0.046

Table 1: The influence of the data points (rows) on the pointwise loss for each data point (columns). The symmetries are due to symmetries in the training data set.

One thing to notice is how the influence of a data point on loss is negative for itself, but positive for other data points. This agrees with the notion that altering the parameters in the direction of the influence function for a data point improves prediction on that point (and others points which are well-predicted by similar parameters), potentially at the cost of others. We also see that the influence of a data point on the loss for itself is especially high.

0.4 Future Work

There are various issues with directly applying the influence function framework to deep learning techniques that are used in practice. The main current technique to adjust for them is by using the *proximal Bregman response function* instead of the basic response function.[1] I have been investigating how this works a bit, and have implemented the proximal Bregman objective in code, but I don't quite have a thorough understanding of them yet. I'd also like to apply these techniques to a more advanced example—maybe a neural network trained on MNIST.

There are a few details about the techniques Anthropic used in their recent paper which I don't have a very good grasp on yet. In particular, when they calculate which training data points were the most influential for some text completion, there is a step where they filter out the vast majority of the training data as uninfluential. I'd like to look more into what's going on there.[2]

I have not really come up with any great ideas for a novel research direction to pursue yet, and I would really like to find something like that to work on next semester, so I think that's a lot of what I will be focusing on in the coming weeks. I might spend some time looking through related robust statistics literature.

References

[1] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger Grosse. If influence functions are the answer, then what is the question?, 2022.

[2] Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions, 2023.

[3] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2020.

[4] Marco Taboga. Parameter of a distribution, 2021. Accessed: 2023-11-21.