Discrete Proof For Models Learned By Gradient Descent Are Kernel Machines

Julien R. Lhermitte

August 29, 2025

1 Introduction

Domingos et. al demonstrates a relation between the similarity of training data samples and the inference point over the history of the changes of parameters for any model learned by gradient descent[2].

Assuming a model $f(x;\omega)$, parametrized by weights ω and taking input x, they arrive at the following interesting result:

$$\lim_{\epsilon\to 0} y = y_0 - \int_{c(t)} \sum_{i=1}^{m} L'(y_i^*, y_i)\vec{\nabla}_\omega f(\vec{x};\vec{\omega}(t)) \cdot \vec{\nabla}_\omega f(\vec{x}_i;\vec{\omega}(t))dt \tag{1}$$

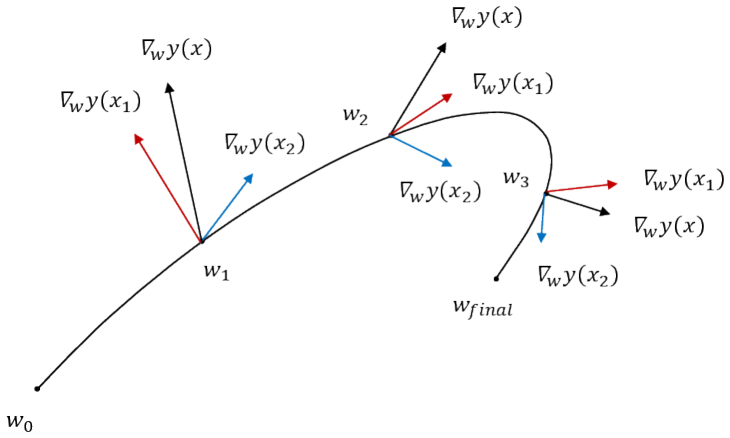

Figure 1: Original visual figure of the path kernel and the similarity from [2].

What this suggests is that no matter where our model starts (y_0), the movement towards the trained model can be approximated as a sum of terms weighted by the similarity between the gradient of the inference point $\vec{x}$ and the training data point. This suggests that for a smooth enough parameter space, that the strongest contributing terms in the sum are the ones where the vector $\vec{x}$ are closest. Intuitively, this is because we can assume some correlation of gradients for neighboring points but no correlation (i.e. random) for other points.

2 The Open Question

The current scheme uses gradient flows [1] and requires that the learning rate ϵ be very small. When training, I believe it may be an open question whether this approximation holds. In particular, it assumes we are not in a local minimum. Being in a local minimum would require us to jump over a max (and reach a point where $\epsilon = 0$).

They mention[2]:

> This is standard in the analysis of gradient descent, and is also generally a good approximation in practice, since the learning rate has to be quite small in order to avoid divergence (e.g., $\epsilon = 10^{-3}$) [3]. Nevertheless, it remains an open question to what extent models learned by gradient descent can still be approximated by kernel machines outside of this regime.

Intuitively this makes a bit of sense. Practically, we observe this with basic LLMs where when giving a very simple prompt asking solve a math specific question with the prefix "You are an MIT student" leads to a higher likelihood better answer than without.

This document aims to discuss what happens when we are outside of this regime. How is this approximation affected? To what extent does it still hold?

3 Discrete Proof

Before we may discuss the limits, we must derive a discrete version of the original proof [2]. We will also find that the key insight that is needed to prove this is that we assume the model can be approximated by a hyperplane (i.e. can be Taylor expanded keeping up to first order derviative terms).

When we undergo gradient descent, we are more or less updating our weights according to a rule:

$$\vec{\omega}_{n+1} = \vec{\omega}_n - \alpha_{n+1}\vec{\nabla}L(\vec{\omega}_n) \tag{2}$$

where α_{n+1} is the learning rate, $\vec{\omega}_n$ the weights at timestep n and L the loss. Note that the loss is implicity dependent upon the training samples (x_i, y_i^*).

We can as in [2] rewrite the gradient of the total loss in terms of a sum over individual gradient losses with respect to the input:

$$\begin{aligned}\vec{\nabla} L(\vec{\omega}_n) &= \sum_j \frac{\partial L}{\partial \omega_j}(\vec{\omega}_n)\hat{\omega_j} \\ &= \sum_{j,m} \frac{\partial L}{\partial y_m}\frac{\partial y_m}{\partial \omega_j}(\vec{\omega}_n)\hat{\omega_j} \\ &= \sum_m \frac{\partial L}{\partial y_m}\vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_n)\end{aligned}$$

where we sum over m training data points with $y_m = f(\vec{x}_m;\vec{\omega}_n)$. Finally, we have:

$$\vec{\omega}_{n+1} - \vec{\omega}_n = -\alpha_{n+1}\sum_m \frac{\partial L}{\partial y_m}\vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_n) \tag{3}$$

The change of the model at each timestep is:

$$\Delta f = f(\vec{x};\vec{\omega}_{n+1}) - f(\vec{x};\vec{\omega}_n) \tag{4}$$

We can rewrite f as a Tayor series, centered at $\vec{\omega}_n$:

$$f(\vec{x};\vec{\omega}_{n+1}) = f(\vec{x};\vec{\omega}_n) + \vec{\nabla}_\omega f(\vec{\omega}_n)\cdot(\vec{\omega}_{n+1} - \vec{\omega}_n) + O(|\omega|^2) \tag{5}$$

The key step here is to assume that our function can be approximated by a hyperplane. We can then ignore the $O(|\omega|^2)$ terms.

If this is true, then we can write the delta as:

$$\begin{aligned}\Delta f &= f(\vec{x};\vec{\omega}_{n+1}) - f(\vec{x};\vec{\omega}_n) \\ &= f(\vec{x};\vec{\omega}_n) - \vec{\nabla}_\omega f\cdot(\vec{\omega}_{n+1} - \vec{\omega}_n) - f(\vec{x};\vec{\omega}_n) \\ &= \vec{\nabla}_\omega f\cdot(\vec{\omega}_{n+1} - \vec{\omega}_n)\end{aligned}$$

If we then replace $\vec{\omega}_{n+1} - \vec{\omega}_n$ with Equation 3, we get:

$$\Delta f = -\alpha_{n+1}\vec{\nabla}_\omega f(\vec{x})\cdot\sum_m \frac{\partial L}{\partial y_m}\vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_n) \tag{6}$$

$$= -\alpha_{n+1}\sum_m \frac{\partial L}{\partial y_m}\vec{\nabla}_\omega f(\vec{x};\vec{\omega}_n)\cdot\vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_n) \tag{7}$$

Finally, the final model will just be the sum of the Δs plus the initial model:

$$f(\vec{x};\vec{\omega_N}) = f(\vec{x};\vec{\omega_0}) + \sum_{k=0}^{N-1} -\alpha_{k+1}\sum_m \frac{\partial L}{\partial y_m}\vec{\nabla}_\omega f(\vec{x};\vec{\omega}_k)\cdot\vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_k)$$

and we end with:

$$f(\vec{x};\vec{\omega_N}) = f(\vec{x};\vec{\omega_0}) + \sum_{k=0}^{N-1} -\alpha_{k+1} \sum_m L'(y_i^*, y_i)\vec{\nabla}_\omega f(\vec{x};\vec{\omega}_k) \cdot \vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_k) \quad (8)$$

which is the discrete version of Equation 1. Note that this is a similar result, except that we find that the learning rate can be variable so long as the function can locally be approximated by a hyperplane.

4 Discussion

Not Quite Kernel Machine With Local Minima (But Still Approximate?)

Equation 8 demonstrates to us that we need simply assume the method is locally approximated by a hyperplane at each time step.

We can quickly see this breaks down when this is not true. When can this happen? This can happen where the slope of the approximate hyperplane is negligible and we must keep higher order terms. This could happen if we happen to be in a local minima or saddle point. In one dimension, this is easy to visualize.

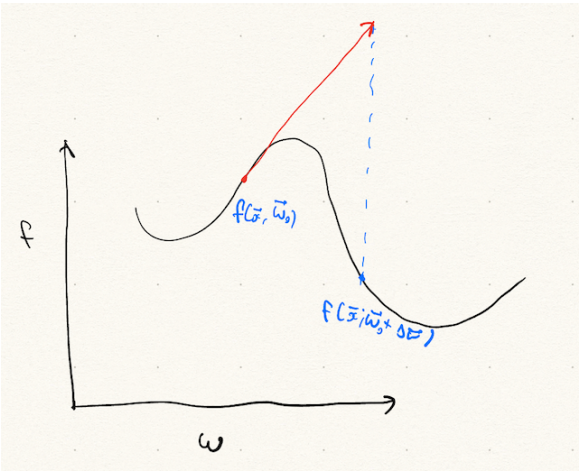

Figure 2: Example of one iteration of gradient descent where the local minima is not the global minima. The only way to get past this minimal is to move to a regime where we hit zero slope, and thus the higher order terms must be included.

See Figure 2. The only way to get to the global minima is to move a $\Delta\vec{\omega}$ where f cannot be approximated by the first order terms in ω. In this case, it means we must account for additional terms. Roughly, we can simply not

ignore the Taylor approximation. Let us group the complex timesteps into T_c. We see we have a sum like:

$$\begin{aligned} f(\vec{x};\vec{\omega_N}) &= f(\vec{x};\vec{\omega_0}) + \\ &\sum_{k=0}^{N-1} -\alpha_{k+1} \sum_m L'(y_i^*, y_i) \vec{\nabla}_\omega f(\vec{x};\vec{\omega}_k) \cdot \vec{\nabla}_\omega f(\vec{x}_m;\vec{\omega}_k) \\ &+ \sum_{\omega_k \in T_c} + O(|\omega_{k+1} - \omega_k|^2) \end{aligned} \tag{9}$$

Whether this terms contributes or not is an open question, and something that will have to be empirically determined. Roughly, I think it's likely related to ratio of local minima encountered to total steps. Training usually involves some sort of regularization which attempts to smooth the space. I would expect this ratio to be small, and likely there to be little contribution.

Kernel Machine When Initialized

Note that a simple solution to this problem would be to simply initialize parameters close to the global minimum. However, this weakens the kernel machines argument. Gradient descent is only a kernel machine if we started near the solution.

5 Follow Up

Janko-dev's notebook demonstrates this empirically. It is possible by tweaking the paths and ensuring we're in a local minimum, we may see if we have agreement or not, and how big the $O(|\omega|^2)$ terms are.

6 Citation

If you find this useful, you can use this BibTex entry:

```
@misc{jlhermitte2025kernelmachine,
  title={Gradient Descent And Kernel Machines},
  author={Julien R. Lhermitte},
  year={2025},
  howpublished={\url{https://jrmlhermitte.github.io/2025/08/08/gradient-descent-and-kernel-m
}
```

References

[1] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient Flows in Metric Spaces and in the Space of Probability Measures*. Lectures in Mathematics ETH Zürich. Birkhäuser, 2. ed edition. OCLC: 254181287.

[2] Pedro Domingos. Every model learned by gradient descent is approximately a kernel machine, 2020.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Book in preparation for MIT Press.