ML HW1

Date: 09/03/2025
Week: 1
Author: Alvin B. Lin
Student ID: 112652040

Problem 1: Simulation for 1-Step GD

1. Consider stochastic gradient descent method to learn the house price model

$$h(x_1, x_2) = \sigma(b + w_1 x_1 + w_2 x_2),$$

where σ is the sigmoid function.

Given one single data point $(x_1,\ x_2,\ y) = (1,\ 2,\ 3)$, and assuming that the current parameter is $\theta^0 = (b,\ w_1,\ w_2) = (4,\ 5,\ 6)$, evaluate θ^1.

Theorem 1: SGD Formula

$$\theta^{n+1} := \theta^n - \alpha \nabla_\theta J(\theta^n)$$

where $J : \mathbb{R}^m \to \mathbb{R}$ is the loss function, $\alpha > 0$ is the learning rate, $n \in \mathbb{N} \cup \{0\}$.

Sol: Choose $J(\theta) = \dfrac{1}{2}\displaystyle\sum_{i=1}^{1}(y^i - h_\theta(x^i))^2$; $h(x) = \sigma(b + w_1 x_1 + w_2 x_2)$, we get

$$\nabla_\theta J(\theta) = \sum_{i=1}^{1}(h(x^i) - y^i) \cdot \nabla_\theta h(x^i); \qquad \nabla_\theta h = \sigma'(x;\theta) \cdot (1,\ x_1,\ x_2)$$

So plug everything in, we get:

$$\theta^1 = \begin{pmatrix}4\\5\\6\end{pmatrix}^T - \alpha\left[\sigma(4 + 5\cdot 1 + 6\cdot 2) - 3\right]\sigma(4+5\cdot 1+6\cdot 2)\left[1 - \sigma(4 + 5\cdot 1 + 6\cdot 2)\right]\begin{pmatrix}1\\1\\2\end{pmatrix}^T,$$

where $\alpha > 0$ is the given learning rate.

Problem 2: General Formula for Sigmoid Function Derivatives

2. (a) Find the expression of

$$\frac{d^k}{dx^k}\sigma$$

in terms of $\sigma(x)$ for $k = 1, \cdots, 3$ where σ is the sigmoid function.

(b) Find the relation between sigmoid function and hyperbolic functions.

Sol: For simplicity, I abbreviate $\sigma(x)$ into σ.

(a) For $k=1$, $\frac{d}{dx}\sigma = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{-(-1)e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{1+e^{-x}} \cdot \frac{1}{1+e^{-x}} = (1-\sigma)\cdot\sigma$
For $k=2$,

$$\begin{aligned}\frac{d^2}{dx^2}\sigma = \frac{d}{dx}\left(\frac{d}{dx}\sigma\right) &= \frac{d}{dx}\left(\sigma(1-\sigma)\right) \\ &= \sigma'\left(1-2\sigma\right) \\ &= \sigma(1-\sigma)(1-2\sigma)\end{aligned}$$

For $k=3$,

$$\begin{aligned}\frac{d^3}{dx^3}\sigma = \frac{d}{dx}\left(\frac{d^2}{dx^2}\sigma\right) &= \frac{d}{dx}\left[\sigma(1-\sigma)(1-2\sigma)\right] \\ &= \sigma'\left[(1-\sigma)(1-2\sigma) - \sigma(1-2\sigma) - 2\sigma(1-\sigma)\right] \\ &= \sigma'(1-6\sigma+6\sigma^2) \\ &= \sigma(1-\sigma)(1-6\sigma+6\sigma^2)\end{aligned}$$

(b) There are 3 basic hyperbolic functions, $\sinh\cdot,\ \cosh\cdot,\ \tanh\cdot$.

> **Lemma 1:** $e^x = \frac{\sigma}{1-\sigma}$
>
> *Proof.* $1-\sigma = 1 - \frac{e^x}{1+e^x} = \frac{1}{1+e^x} \implies \frac{\sigma}{1-\sigma} = \frac{\frac{e^x}{1+e^x}}{\frac{1}{1+e^x}} = e^x$ ☐

For $\sinh x$, $\sinh x = \frac{e^x - e^{-x}}{2} = \frac{\frac{\sigma}{1-\sigma} - \frac{1-\sigma}{\sigma}}{2} = \frac{\sigma^2-(1-\sigma)^2}{2\sigma(1-\sigma)} = \frac{2\sigma-1}{2\sigma(1-\sigma)}$

For $\cosh x$, $\cosh x = \frac{e^x + e^{-x}}{2} = \frac{\frac{\sigma}{1-\sigma} + \frac{1-\sigma}{\sigma}}{2} = \frac{\sigma^2+(1-\sigma)^2}{2\sigma(1-\sigma)} = \frac{2\sigma^2-2\sigma+1}{2\sigma(1-\sigma)}$

For $\tanh x$, $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}} = \frac{2-(1+e^{-2x})}{1+e^{-2x}} = 2\sigma(2x)-1$

A more straightforward interpretation can be determined by the graph:

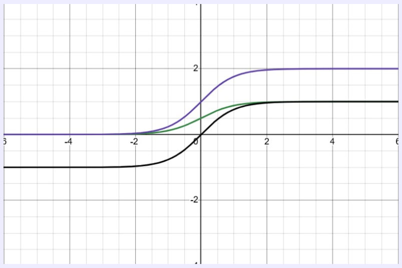

Figure 1: Graph of $\sigma(2x),\ 2\sigma(2x), 2\sigma(2x)-1$ and $\tanh x$

By the graph, the green line is $\sigma(2x)$, the purple one is $2\sigma(2x)$, the black one is $2\sigma(2x)-1$, which is also $\tanh x$.

My Question 1: Why Isn't Backprop Done Layer-by-Layer?

Given the interconnected nature of a neural network, it seems intuitive to **optimise each layer sequentially**, from the output layer inward. However, backpropagation computes the **full gradient** for all layers in one pass. Could you elaborate on why optimising one layer at a time would be inefficient or potentially lead to a suboptimal solution? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 1

This seems logical, but it has a major **flaw**: The layers are **interconnected**. When you adjust the weights of the final layer, you're doing so based on the current, un-optimised outputs of the previous layer. Once you go back and change the weights of that earlier layer, the output of that layer will change, and therefore the "optimal" weight adjustments you made to the final layer are no longer correct. You would have to **constantly go back and forth**, re-optimising each layer, which is a slow and computationally expensive process that may never truly converge to a good solution. ——— (Answer from `Gemini`)

My Question 2: Why Isn't Learning Rate Dynamic?

Standard Gradient Descent uses a **fixed** learning rate, which presents challenges such as **slow convergence** or **overshooting the minimum**. Given this, why is it not standard practice to **dynamically** adjust the learning rate during training? Furthermore, could we conceptualise the learning rate itself as a **hyperparameter** to be optimised, perhaps even using a meta-optimisation approach like a nested gradient descent, to make it more adaptable to the loss landscape? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 2

Your intuition regarding a dynamic learning rate is a **cornerstone of modern optimisation theory** in deep learning. The limitations of a fixed learning rate—namely, the **trade-off** between rapid **convergence** and **stability**—have led to the development of a class of algorithms known as **adaptive learning rate methods**.

Instead of a **fixed** learning rate (η), these optimisers dynamically adjust the step size for each parameter based on the history of its gradients. This approach effectively addresses the issues you raised:

1. **Addressing Suboptimal Initial Learning Rates**: Adaptive optimisers are far more resilient to poorly chosen initial learning rates. They can start with a **large** step size for fast initial progress and automatically reduce it as the model approaches a minimum, preventing overshooting.
2. **Optimising for Different Parameters**: This method provides **unique** learning rates for each parameter. For instance, a weight with a consistently high gradient history will have its learning rate scaled down, while a weight with small, infrequent gradients will have its learning rate increased. This allows the model to fine-tune some parameters while making larger updates to others, leading to more efficient convergence.

> This is fundamentally different from a meta-optimisation approach (like performing a nested gradient descent on the learning rate itself), which would be computationally prohibitive. Instead, adaptive optimisers compute the learning rate updates as a function of the primary network gradients, **a far more computationally efficient method**.
>
> Prominent examples of these algorithms include **AdaGrad**, **RMSprop**, and the highly popular **Adam** (Adaptive Moment Estimation), which combines the benefits of adaptive learning rates with momentum to accelerate training. Thus, the practice you proposed is not only viable but has become the de facto standard for training most neural networks.
>
> This version uses more formal language, explicitly names the class of algorithms, provides a clear list of benefits, and directly addresses the "meta-optimisation" part of the refined question, clarifying why a direct nested gradient descent is not the chosen approach.
>
> ——— (Answer from `Gemini`)