HW 6

Dylan Doby

1/21/2024

What is the difference between gradient descent and *stochastic* gradient descent as discussed in class? (*You need not give full details of each algorithm. Instead you can describe what each does and provide the update step for each. Make sure that in providing the update step for each algorithm you emphasize what is different and why.*)

Student Input

The primary difference between gradient descent (GD) and stochastic gradient descent (SGD) lies in how they compute the gradient during the optimization process. Gradient descent calculates the gradient of the loss function using the entire dataset at each iteration. Its update step is defined as:

$$\theta_{i+1} = \theta_i - \alpha \nabla f(\theta_i, X, Y)$$

where

$$\nabla f(\theta_i, X, Y)$$

represents the gradient computed over the entire dataset, and   is the learning rate. This approach ensures stable and precise updates but can be computationally expensive, especially for large datasets, as every iteration involves processing all data points.

In contrast, stochastic gradient descent approximates the gradient by using a randomly selected subset of the data, or even a single data point, at each iteration. The update step for SGD is:

$$\theta_{i+1} = \theta_i - \alpha \nabla f(\theta_i, X_i', Y_i')$$

where

$$\nabla f(\theta_i, X_i', Y_i')$$

is the gradient computed from the subset

$$(X_i', Y_i')$$

, and   remains the learning rate. By only processing a small portion of the data at a time, SGD reduces computational cost significantly and allows for faster updates, making it particularly efficient for large-scale datasets. However, this efficiency comes at the cost of stability, as the updates introduce noise, leading to a less smooth path to convergence compared to GD.

While GD is ideal for smaller datasets or scenarios requiring high precision, SGD is better suited for large-scale datasets and online learning applications where computational efficiency is more of a priority. The randomness in SGD also helps the algorithm escape local minima, though it results in noisier convergence. This distinction makes SGD a popular choice in many machine learning contexts, especially when training models on extensive data.

Consider the `FedAve` algorithm. In its most compact form we said the update step is $\omega_{t+1} = \omega_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla F_k(\omega_t)$. However, we also emphasized a more intuitive, yet equivalent, formulation given by $\omega_{t+1}^k = \omega_t - \eta \nabla F_k(\omega_t); w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$.

Prove that these two formulations are equivalent.

(*Hint: show that if you place ω_{t+1}^k from the first equation (of the second formulation) into the second equation (of the second formulation), this second formulation will reduce to exactly the first formulation.*)

Student Input

To prove that the two formulations of the `FedAve` algorithm are equivalent, we start by substituting ω_{t+1}^k from the first equation in the second formulation into the second equation in the same formulation.

Given:

1. **First formulation**:

$$\omega_{t+1} = \omega_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla F_k(\omega_t)$$

2. **Second formulation**:

   - First equation:

$$\omega_{t+1}^k = \omega_t - \eta \nabla F_k(\omega_t)$$

   - Second equation:

$$\omega_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \omega_{t+1}^k$$

Proof:

Start with the second equation of the second formulation:

$$\omega_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \omega_{t+1}^k$$

Substitute $\omega_{t+1}^k = \omega_t - \eta \nabla F_k(\omega_t)$ into this equation:

$$\omega_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \left(\omega_t - \eta \nabla F_k(\omega_t)\right)$$

Distribute $\frac{n_k}{n}$ across the terms in the summation:

$$\omega_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \omega_t - \sum_{k=1}^{K} \frac{n_k}{n} \eta \nabla F_k(\omega_t)$$

Factor out ω_t from the first summation:

$$\omega_{t+1} = \omega_t \sum_{k=1}^{K} \frac{n_k}{n} - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla F_k(\omega_t)$$

Since $\sum_{k=1}^{K} \frac{n_k}{n} = 1$ (because the weights $\frac{n_k}{n}$ represent proportions that sum to 1):

$$\omega_{t+1} = \omega_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla F_k(\omega_t)$$

This is exactly the first formulation. Thus, the two formulations are equivalent.

Now give a brief explanation as to why the second formulation is more intuitive. That is, you should be able to explain broadly what this update is doing.

Student Input

The second formulation of the `FedAve` algorithm is more intuitive because it breaks the update process into two clear and interpretable steps:

1. **Local Updates**: Each client k computes its own updated model ω_{t+1}^k based on its local data. This is represented by the equation:
$$\omega_{t+1}^k = \omega_t - \eta \nabla F_k(\omega_t)$$
This step shows that each client individually performs gradient descent on its local loss function $F_k(\omega_t)$ using only its own data.

2. **Global Aggregation**: After all clients compute their local updates, the central server aggregates these updates into a single global update. This is represented by:
$$\omega_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \omega_{t+1}^k$$
This step shows that the global update is a weighted average of the local updates, where the weights $\frac{n_k}{n}$ are proportional to the amount of data held by each client. This ensures that larger datasets have a greater influence on the global model.

Intuition Behind the Update

Broadly, the two-step process of the FedAve algorithm reflects a real-world collaborative system. In the first step, each client independently updates its model by learning from its own local environment through local optimization. In the second step, the central server aggregates these individual updates to produce a global model, effectively combining the contributions of all clients. This formulation simplifies understanding by clearly distinguishing the roles of the clients, which perform localized learning, and the server, which performs the aggregation. Additionally, it shows how the global model reflects the contributions of each client, weighted by the size and relevance of their local data. This structure highlights the cooperative nature of the process while giving larger datasets a proportionally greater influence on the global model.

Prove that randomized-response differential privacy is ϵ-differentially private.

Student Input

Definition: -Differential Privacy

For $\epsilon > 0$, and a randomized algorithm A, A is said to be ϵ-differentially private if and only if, for any two datasets D_1 and D_2 differing in exactly one element, and for any subset $S \subseteq \text{im}(A)$:

$$\frac{P[A(D_1) \in S]}{P[A(D_2) \in S]} \leq e^{\epsilon}.$$

The intuition is that smaller ϵ values provide stronger privacy guarantees. As $\epsilon \to 0$, the right-hand side approaches 1, which implies that D_1 and D_2 are indistinguishable. This ensures that the output of A does not reveal whether an individual's data is included in the dataset.

Fact:

Randomized Response Differential Privacy is ϵ-Differentially Private where $\epsilon = \ln(3)$.

Proof:

Observe that D and $S \in \{\text{Yes, No}\}$. Consider the case $S = \text{Yes}$.

$$\frac{P[A(\text{Yes}) = \text{Yes}]}{P[A(\text{No}) = \text{Yes}]} = \frac{P[\text{Output} = \text{Yes} \,|\, \text{Input} = \text{Yes}]}{P[\text{Output} = \text{Yes} \,|\, \text{Input} = \text{No}]} = \frac{3/4}{1/4} = 3 = e^{\ln(3)}.$$

Similarly:

$$\frac{P[A(\text{No}) = \text{Yes}]}{P[A(\text{Yes}) = \text{Yes}]} = \frac{1}{3}.$$

Thus:

$$\frac{P[A(D_1) \in S]}{P[A(D_2) \in S]} \leq e^{\ln(3)}.$$

The randomized response mechanism satisfies ln(3)-Differential Privacy.

Define the harm principle. Then, discuss whether the harm principle is *currently* applicable to machine learning models. (*Hint: recall our discussions in the moral philosophy primer as to what grounds agency. You should in effect be arguing whether ML models have achieved agency enough to limit the autonomy of the users of said algorithms.*)

Student Input

The Harm Principle, introduced by John Stuart Mill, says that personal autonomy should only be restricted to prevent harm to others. In simpler terms, people should be free to make their own decisions as long as those decisions don't negatively impact anyone else. Mill focuses on protecting individual freedom, but with the understanding that it has limits when someone else's well-being is at risk.

When thinking about how this applies to machine learning (ML) models today, the key question is whether these models have enough "agency" to restrict human autonomy. Agency, in this case, is the ability to act intentionally and be morally responsible for those actions. Even though ML models, like language models or predictive algorithms, can do some pretty incredible things, they don't have the capacity to act on their own, make moral judgments, or even understand the consequences of their outputs. Because of this, ML models themselves don't violate the harm principle. They're tools, not independent agents. So, while ML models themselves don't cause harm in the way a person might, the harm can come from the way humans design or use them.

That being said, ML models can still lead to real harm, even if they aren't the ones directly "doing" it. For example, biased algorithms can reinforce discrimination, black-box systems can undermine people's autonomy, and social media algorithms can amplify harmful behaviors or misinformation. But these outcomes usually reflect poor design, bad oversight, or careless deployment by humans rather than something the ML models themselves are responsible for. So, in this sense, it's the developers, companies, and regulators behind these tools who should be held accountable under the harm principle—not the algorithms.

The harm principle is also tied to the idea of informed consent. Mill believed that if people fully understand the risks and still willingly consent, the resulting harm isn't unjust. For example, if someone agrees to share their personal data with an app, knowing it might be used by advertisers or insurers in ways they don't like, Mill would say they've made an informed choice, and any harm isn't a violation of their autonomy. The problem arises when people aren't informed, don't understand the risks, or are coerced. Without clear and voluntary consent, harm becomes harder to justify.

There's also a case to be made for rare exceptions where autonomy might justifiably be limited to prevent significant harm. For instance, if an ML system identifies a major public safety threat, temporary restrictions on individual freedom (like using personal data without consent) might be defensible. Still, these exceptions should be rare and carefully managed, as preserving autonomy is usually the default.

Overall, while ML models themselves don't have the agency needed to directly invoke the harm principle, the way we design, implement, and oversee them absolutely does. Developers and organizations would need to focus on minimizing harm, being transparent, and respecting user autonomy to meet the ethical obligations outlined by Mill.