Notes with Examples

1. Gradient Computation in PyTorch (`y.backward()`)

When you call `y.backward()` on a tensor y that depends on another tensor x, the `.grad` attribute of x stores the gradient of y with respect to x. This shows how much each component of x influences the changes in y.

Example:

```
import torch

x = torch.tensor([2.0, 3.0], requires_grad=True)  # x requires gradients
y = x[0]**2 + 3 * x[1]  # A function of x
y.backward()  # Compute gradients

print(x.grad)  # Gradients of y with respect to x
```

Output:

$$\text{tensor}([4.0, 3.0])$$

For $x[0]$, the gradient is $\frac{\partial y}{\partial x[0]} = 4.0$.
For $x[1]$, the gradient is $\frac{\partial y}{\partial x[1]} = 3.0$.

2. Difference Between Probability and Statistics

- **Probability:** A theoretical measure predicting the likelihood of an event based on assumptions or models.
- **Statistics:** An empirical measure derived from data or observations.

Example:

- **Probability:** Predicting the chance of rolling a 6 on a fair die is $\frac{1}{6}$, as the process is theoretical.
- **Statistics:** Rolling a die 100 times and observing that a 6 appears 18 times gives an empirical probability of $\frac{18}{100} = 0.18$.

3. Frequency in Statistics

Frequency is the count of an event's occurrence relative to the total number of trials or observations.

Example: Toss a coin 10 times, and it lands on heads 6 times. The frequency of heads is:

$$\text{Frequency of heads} = \frac{\text{Number of heads}}{\text{Total tosses}} = \frac{6}{10} = 0.6$$

4. Law of Large Numbers and Error Rates

The **Law of Large Numbers** states that as sample size n increases, the observed mean or property approaches the true mean, reducing error rates. The error decreases by a factor of $\frac{1}{\sqrt{n}}$.

Example: Suppose the true proportion of heads in a fair coin toss is 0.5.

- For $n = 10$: The observed proportion might vary significantly (e.g., 0.7, 0.3).
- For $n = 1,000$: The observed proportion will be closer to 0.5 (e.g., 0.49, 0.51).

This reduction in variability/error follows $\frac{1}{\sqrt{n}}$. For $n = 10$, the error might be ± 0.1, while for $n = 1,000$, the error reduces to ± 0.01.

5. Stochastic Gradient Descent (SGD): Advantages and Drawbacks

The extreme case of optimization is to consider only a single example at a time and update the model based on that observation. This approach is called **Stochastic Gradient Descent (SGD)**. It can be an effective strategy for large datasets, as noted by Bottou (2010).

Advantages of SGD

- **Scalability:** Processes one observation at a time, making it suitable for massive datasets where full-batch processing is computationally infeasible.
- **Efficiency:** Can converge faster in some cases due to its ability to "escape" saddle points and poor local minima.

Drawbacks of SGD

1. **Computational inefficiency:**
   - Modern processors are optimized for operations such as **matrix–vector multiplications**, which are much faster than performing an equivalent number of **vector–vector operations**.
   - Processing one sample at a time leads to high memory access overhead, making it up to an order of magnitude slower than using a batch.
2. **Statistical inefficiency:**
   - Certain layers, such as **batch normalization**, require multiple observations to perform effectively.
   - Lack of diversity in a single observation might lead to noisy or suboptimal gradient updates.

Summary: While SGD is powerful and widely used, especially for large-scale learning problems, it is important to consider its computational and statistical limitations. Techniques like **mini-batch gradient descent** often balance the trade-offs between SGD and full-batch gradient descent.

6. Hyperparameters: Mini-batch Size and Learning Rate

Frequently, the **mini-batch size** and **learning rate** are user-defined. Such tunable parameters that are not updated within the training loop are referred to as **hyperparameters**.

Key Points on Hyperparameters

- **Mini-batch Size:** Determines the number of samples processed before the model updates its parameters. A larger batch size provides a more stable gradient but requires more computational resources.
- **Learning Rate:** Controls the size of the steps taken during optimization. A small learning rate results in slow convergence, whereas a large learning rate may cause the model to overshoot the optimal solution.

Tuning Hyperparameters:

- Hyperparameter tuning often involves trial and error or automated search techniques such as grid search or random search.
- The choice of hyperparameters can significantly affect model performance and training efficiency.