Lecture 9: Creating Input-Target Data Pairs

1. The Goal: Data Pre-processing for Training

This lecture bridges the gap between **tokenization** (Lecture 8) and **vector embeddings** (Lecture 10).

- **The Challenge:** Unlike image classification (where inputs are images and outputs are labels like "cat" or "dog") or regression (where inputs are features and outputs are prices), defining the input and output for an LLM is not immediately obvious.
- **The Objective:** We must structure raw text into **Input-Target pairs** to train the model to predict the next word.

2. Conceptual Framework: Auto-Regression

The mechanism used to train LLMs is called **Auto-regression**, where the output of one step becomes the input for the next.

- **The Process:**
  - The model looks at a sequence of words (Input) and predicts the very next word (Target).
  - Everything past the target word is **masked** (hidden) during that specific training step.
- **Example Iterations:** Consider the sentence: *"LLMs learn to predict one word at a time"*.
  - **Iteration 1:** Input: `LLMs` $\rightarrow$ Target: `learn`.
  - **Iteration 2:** Input: `LLMs learn` $\rightarrow$ Target: `to`.
  - **Iteration 3:** Input: `LLMs learn to` $\rightarrow$ Target: `predict`.
- **Self-Supervised Learning:** This process is unsupervised (or self-supervised) because we do not need human-created labels; the structure of the sentence itself provides the labels.

3. Constructing Inputs (X) and Targets (Y)

To implement this in code, we create two tensors, X (Input) and Y (Target), using a **sliding window approach**.

- **The Shift:** The target tensor Y is essentially the input tensor X shifted by one position.
  - If Input X is `, the Target $Y$ is`.
- **Context Size (Context Length):**
  - This defines how many tokens the model looks at to make a prediction.
  - If the context size is 4, the input will contain 4 tokens.
  - **Multiple Predictions per Batch:** A single input-target pair with a context size of 4 actually contains **four** distinct prediction tasks:
    1. Input: `` $\rightarrow$ Target: `2`
    2. Input: `` $\rightarrow$ Target: `3`
    3. Input: `` $\rightarrow$ Target: `4`

4. Input: `` $\rightarrow$ Target: `5`.

4. Implementing the Data Loader (Python & PyTorch)

To handle large datasets efficiently, we use PyTorch's `Dataset` and `DataLoader` classes to create tensors and manage batches.

A. The Dataset Class (`GPTDatasetV1`)

- **Tokenization:** First, the entire raw text (e.g., "The Verdict" short story) is encoded into token IDs using the Byte Pair Encoder (BPE),.
- **Chunking:** The text is divided into chunks based on the `max_length` (context size).
- **The `__getitem__` method:** This function retrieves a specific row of data based on an index. It returns the input tensor row and the corresponding target tensor row.

B. The Sliding Window & Stride The **Stride** determines how positions move when creating the next batch.

- **Stride = 1:** The window slides by only one token.
  - *Result:* High overlap between batches. (e.g., Batch 1: "In the heart of", Batch 2: "the heart of the").
  - *Risk:* Can lead to overfitting due to repetitive data.
- **Stride = Context Length:** The window slides by the full length of the context.
  - *Result:* No overlap. (e.g., Batch 1: "In the heart of", Batch 2: "the city stood the").
  - *Benefit:* Utilizes the dataset fully without repetition.

C. The DataLoader Class This handles the logistics of feeding data into the model during training.

- **Batch Size:** The number of data samples the model processes before updating its parameters (weights).
  - *Small Batch Size:* Faster parameter updates but noisier.
  - *Large Batch Size:* More stable updates but takes longer per update.
- **Num Workers:** Allows parallel processing using multiple CPU threads to speed up data loading.
- **Drop Last:** A parameter often set to `True` to drop the final batch if it is shorter than the specified batch size to prevent loss spikes.

5. Summary of Tensors

The final output of the Data Loader is a set of tensors ready for training:

- **Input Tensor:** Dimensions are `[Batch Size, Context Length]`.
- **Target Tensor:** Dimensions are `[Batch Size, Context Length]` (shifted by one).
- The model uses these tensors to learn how to predict the next token based on the provided context history.