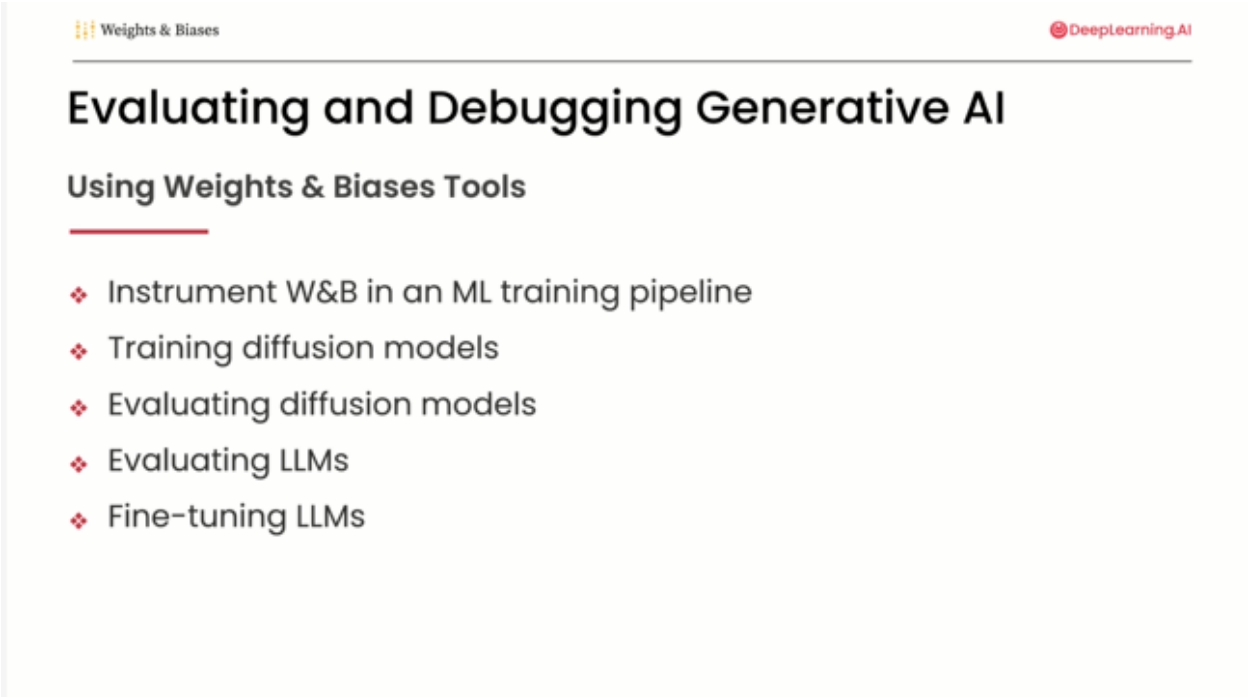
Evaluating and Debugging Generative AI



[**Lesson 1: Introduction to Weights & Biases (wandb) for ML Monitoring and Debugging**]
[**Lesson 2: Training Diffusion Models with "wandb"**]
[**Lesson 3: Comparing Diffusion Model Outputs and Evaluating LLM Models**]
[**Lesson 4: Evaluating Large Language Models (LLMs)**]
[**Lesson 5: Fine-Tuning Large Language Models (LLMs)**]

Weights & Biases

DeepLearning.AI

Weights & Biases MLOps Portfolio

Tools for Machine Learning Practitioners

W&B Models

- Experiments — Experiment tracking
- Models — Model lifecycle management
- Automations — ML workflow automation
- Launch — Packaged ML workflow jobs
- Sweeps — Hyperparameter optimization

W&B Prompts

- Prompts — LLMOps and prompt engineering

W&B Monitoring

- Monitoring — Online ML Production Monitoring

W&B Platform

- Artifacts — Data and model versioning
- Tables — Interactive data visualization
- Reports — Collaborative dashboards
- Weave — Platform-extension toolkit

Weights & Biases

DeepLearning.AI

W&B MLOps Platform

Integrated into every popular ML framework

PyTorch | TensorFlow | Keras | PyTorch Lightning | JAX | scikit-learn

spaCy | HUGGING FACE | LangChain | LlamaIndex | OpenAI

Runs on every cloud or in your own infra

aws | Google Cloud | Microsoft Azure | NVIDIA

Lesson 1: Introduction to Weights & Biases (wandb) for ML Monitoring and Debugging

Overview
- Introduction to instrumenting weights and biases in ML training code.
- "wandb" for monitoring, debugging, and evaluating ML pipelines.

Benefits of "wandb"
- Real-time monitoring of metrics, CPU, and GPU usage.
- Version control for code and model checkpoints.
- Centralized, interactive dashboard for visualization.
- Configurable reports for model evaluation and bug discussion.

Incorporating "wandb"
- Install "wandb" with `pip install wandb`.
- Import "wandb" library and prepare hyperparameters in a config object.

Initiating a "wandb" Run
- Call `wandb.init` with project name and config to start a run.
- A "run" corresponds to a machine learning experiment.

Logging Metrics
- Use `wandb.log` to track and visualize metrics during training.
- Log metrics when needed for analysis.

Training Process Integration
- Incorporate "wandb" into your training loop.
- Log metrics using `wandb.log` during training.
- Record validation metrics at end of each epoch.

Finishing a Run
- Optionally, call `wandb.finish` to end a "wandb" run.
- Especially recommended when using notebooks.

Training Sprite Classification Model Example
- Import necessary libraries, including "wandb".
- Define a simple classifier model with linear layers.

Training Function Modification
- Modify training function to include "wandb" logging.
- Call `wandb.init` and pass project name and config.
- Log metrics using `wandb.log` during training.

"wandb" Cloud Platform and Login
- Use "wandb" Cloud Platform for this course.
- Log in to "wandb" using personal account and API key.
- Login enables experiment tracking and result saving.

Training the Model
- Run training code to observe progress.
- Data is logged to "wandb" server for result storage.

Visualization and Comparison on "wandb"
- Access "wandb" project page to view results.
- View training loss and validation metrics.
- Compare different runs for performance analysis.

Hyperparameter Tuning
- Experiment with hyperparameters to improve model.
- Modify hyperparameters for better performance.

Comparing Experiments
- Use project page to compare experiment results.
- Hover over runs to view training curves.
- Utilize runs table for side-by-side metric and hyperparameter comparison.

Filtering and Sorting Experiments
- Apply filters to focus on specific runs.
- Sort runs by metrics to identify top performers.

Detailed Run Overview
- View details of specific run in detail view.
- Access Git repo and commit hash for code reference.
- Capture uncommitted changes in diff patch.
- Config captures settings for reproducibility and communication.

Conclusion and Next Lesson
- Summary of lesson content.
- Teaser for next lesson on generative AI model training with "wandb."

Lesson 2: Training Diffusion Models with "wandb"

Introduction
- Building upon previous lesson's model instrumentation.
- Focus on training a diffusion model using "wandb."

Diffusion Models Overview
- Diffusion models are denoising models.
- Trained to remove noise from images, not generate them.
- Noise added to images following a scheduler, model predicts noise.
- Samples generated by removing noise iteratively.

Importance of Telemetry
- Metrics like loss curve are important but might not reflect image quality.
- Regularly sample from the model during training for better insight.
- Image quality might improve even when loss plateaus.

Logging Samples and Model Checkpoints
- Uploading samples to "wandb" for visualization.
- Saving model checkpoints for organization.

Notebook Overview
- Using Deeplearning.ai's diffusion model training notebook.
- Importing relevant libraries and "wandb."
- Creating an account for result tracking (or anonymous logging).

Environment Variables Setup
- Defining paths for model and checkpoint storage.
- Utilizing CUDA GPU if available.

Hyperparameters Setup
- Using a simple namespace to set varying hyperparameters.
- Importing "ddpm" noise scheduler and sampler.

Creating the Neural Network
- Creating the neural network to be trained.

Data Loading and Optimization
- Using a sample dataset.
- Creating a data loader and setting up an optimizer.

Training Loop Setup
- Choosing noise for sampling.

- Preparing for the training loop.

Training Phase
- Initializing a "wandb" run to track training.
- Passing project name, classification, and job type.
- Logging configuration and passing `wandb.config`.

Standard Training Loop
- Running forward and backward passes for several epochs.
- Logging metrics to "wandb," including loss and learning rate.

Saving Model Checkpoints
- Saving model checkpoints every few epochs.
- Using "wandb" artifact to version and store files.

Image Logging
- Logging sample images using "wandb.log" and "wandb.image."

Finishing the Run
- Calling `wandb.finish` to conclude the run.

Visualizing Training Progress
- Viewing loss curve and sample images in "wandb" workspace.

Model Registry
- Linking the trained model to the Model Registry.
- Centralized location for best model versions.
- Lineage tracking and Git commit reference.

Conclusion and Next Lesson
- Recap of lesson content.
- Teaser for the next lesson on sampling a diffusion model.

Lesson 3: Comparing Diffusion Model Outputs and Evaluating LLM Models

Introduction
- Comparing diffusion model outputs in this lesson.
- Starting with the model trained in the previous lesson.

Model Registry Overview
- Model Registry as central system for machine learning models.
- Manages lifecycle from staging to production.
- Detailed lineage tracking during training, evaluation, production.
- Automates downstream tasks for efficiency.

Tables for Comparison and Evaluation
- Using tables for data logging, query, analysis.
- Create a table, define columns, update rows, log with "wandb.log."

Pulling Model from Registry
- Pulling model from Model Registry using "wandb.Api."
- Retrieving model and run information.

Loading Model Weights
- Loading model weights from artifact.
- Recreating model using original parameters.

Diffusion Sampler Setup
- Setting up "ddpm" diffusion sampler.
- Defining fixed noises and context vector.

Comparing "ddpm" and "ddim" Samplers
- Importing another sampler, "ddim," for comparison.
- Generating samples using both samplers.

Creating a Visual Table
- Constructing a table for visual comparison.
- Adding rows with images, class name, input noise.

Logging the Table
- Calling "wandb.init" with project name and job type.
- Setting table name and logging it with "wandb.log."

Visualizing Comparison Results
- Opening run to view the uploaded table.

- Exploring rows with sample images and information.

Grouping and Filtering Samples
- Grouping images by class for side-by-side comparison.
- Hiding unnecessary columns for better visualization.

Creating and Sharing a Report
- Creating a report with the sample table.
- Adding context and notes to explain findings.
- Publishing report to make it available for colleagues.

Conclusion and Next Lesson
- Summary of lesson content.
- Teaser for the next lesson on evaluating an LLM model.

Lesson 4: Evaluating Large Language Models (LLMs)

Introduction
- Focus on evaluating large language models in this lesson.
- Explore three examples to understand evaluation and debugging.

Example 1: Using API for LLM Evaluation
- Designing system and user prompts.
- Calling OpenAI API using chat completion.
- Parsing and logging results with "wandb" tables.

Example 2: Tracing LLM Chains with a Tool Called Tracer
- Creating a custom LLM chain.
- Using Tracer for tracking and debugging complex chains.
- Illustrating chain concept with World Picker and Name Generator.

Example 3: LLM Chains with LangChain Agents
- Introducing LangChain agent concept.
- Using WorldPicker and NameValidator tools.
- Running queries and analyzing results.

Conclusion and Teaser
- Summarizing lesson content.
- Teasing the next lesson on fine-tuning LLMs.

Lesson 5: Fine-Tuning Large Language Models (LLMs)

Introduction
- Discussing the need for fine-tuning or training new LLMs.
- Emphasizing the importance of debugging and evaluation during the process.

Training LLMs from Scratch
- Training LLMs from scratch is time-consuming and resource-intensive.
- Monitoring training progress, metrics, and using checkpoints.
- Utilizing Weights and Biases dashboard for insights and checkpoints.

Fine-Tuning LLMs
- Fine-tuning is more economical and feasible.
- Careful evaluation process is still crucial.
- Tailoring evaluation strategies based on intended model usage.

Fine-Tuning Example with Hugging Face
- Fine-tuning a small language model (TinyStories) on character backstories.
- Importing necessary libraries and logging in.
- Pulling dataset from Hugging Face hub and examining its structure.
- Preparing dataset for training by tokenizing and padding.
- Creating a causal language model for autoregressive language modeling.
- Setting up training arguments, streaming metrics to Weights and Biases.

Fine-Tuning Process
- Starting a new Weights and Biases run for training.
- Monitoring training progress with live metrics.
- Generating samples from the trained model and evaluating results.
- Using qualitative evaluation and potential metrics like unique words.

Conclusion and Next Lesson
- Recap of lesson content.
- Introduction to the next lesson on deploying LLMs to production.