**PART ONE**

**1. Short Answer Questions**

**Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?**

**TensorFlow** and **PyTorch** are both powerful deep learning frameworks, but they differ in computation style, usability, and deployment capabilities.

1. **Computation Graphs:**
   * **TensorFlow** originally used *static computation graphs* (defined before execution). This improves performance during deployment but can be less flexible during model development.
   * **PyTorch** uses *dynamic computation graphs*, meaning the graph is built as operations run. This allows immediate feedback and easier debugging, making it more intuitive for researchers.
2. **Ease of Use:**
   * **PyTorch** feels more “Pythonic”; it integrates smoothly with standard Python libraries and control flow, which makes experimentation easier.
   * **TensorFlow**, especially with **Keras**, provides high-level APIs that simplify model creation, making it great for production and large-scale applications.
3. **Deployment:**
   * **TensorFlow** integrates seamlessly with **TensorFlow Serving**, **TensorFlow Lite**, and **TensorFlow.js**, offering robust deployment to servers, mobile devices, and browsers.
   * **PyTorch** has improved deployment options through **TorchServe** and **ONNX**, but TensorFlow remains more mature for production environments.

**When to Choose:**

* Choose **TensorFlow** for large-scale applications and production environments.
* Choose **PyTorch** for research, prototyping, and when you need dynamic graph flexibility.

**Q2: Describe two use cases for Jupyter Notebooks in AI development.**

1. **Model Prototyping and Experimentation:**  
   Jupyter Notebooks allow developers to write and test code interactively, visualize outputs immediately, and adjust model parameters on the fly. This is ideal for experimenting with machine learning algorithms, tuning hyperparameters, and visualizing model performance using plots or tables.
2. **Data Analysis and Visualization:**  
   Data scientists use Jupyter to clean, explore, and visualize datasets before model training. With integrations like Matplotlib, Seaborn, and Pandas, it’s easy to generate visual insights (e.g., correlation heatmaps, feature distributions) that inform the design of AI models.

Jupyter Notebooks are also widely used for teaching and collaborative documentation, since they combine code, markdown explanations, and outputs in one file.

**Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?**

While basic Python string operations can handle simple text tasks (like splitting or finding words), **spaCy** provides **advanced linguistic processing capabilities** that go far beyond simple text manipulation.

* **Tokenization & Linguistic Features:**  
  spaCy can automatically break text into tokens (words, phrases, symbols) and annotate them with linguistic information such as part-of-speech tags, lemmas, and syntactic dependencies.
* **Named Entity Recognition (NER):**  
  It identifies and classifies entities such as names, organizations, and locations something not possible with simple string methods.
* **Efficiency and Scalability:**  
  spaCy is written in Cython and optimized for speed, allowing it to process large volumes of text efficiently.
* **Pretrained Models and Pipelines:**  
  spaCy provides pretrained models for multiple languages, enabling out-of-the-box NLP tasks such as sentence segmentation, text classification, and dependency parsing.

**In summary:** spaCy transforms raw text into structured, meaningful data making NLP tasks faster, more accurate, and more powerful than what could be achieved with basic string operations.

**2. Comparative Analysis**

**Q4: Comparative Analysis: Scikit-learn vs. TensorFlow**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Scikit-learn** | **TensorFlow** |
| **Target Applications** | Classical machine learning algorithms (e.g., SVM, Decision Tree, Random Forest, KNN). Best for small to medium datasets. | Deep learning and neural networks (CNNs, RNNs). Best for large-scale and high-dimensional data. |
| **Ease of Use** | Extremely user-friendly with simple APIs (fit, predict). Great for beginners. | Steeper learning curve; requires understanding of tensors, layers, and training loops. |
| **Community Support** | Large community, especially for classical ML and academic use. | Very active community, heavily supported by Google; strong in industry and production environments. |
| **Visualization & Tools** | Integrates well with Pandas, NumPy, and Matplotlib. | Integrates with TensorBoard for training visualization and model debugging. |
| **Deployment** | Limited built-in deployment features (requires external libraries). | Offers complete deployment ecosystem (TensorFlow Serving, Lite, and JS). |

**Summary:**  
Use **Scikit-learn** for quick, interpretable machine learning experiments and traditional algorithms. Use **TensorFlow** for building and deploying deep neural networks that require scalability and GPU acceleration.