📘 Part 1: Theoretical Understanding

1. Short Answer Questions

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

Answer:
TensorFlow and PyTorch are two of the most widely used deep learning frameworks today. While they serve similar purposes, they differ significantly in how they operate and are applied.

- **TensorFlow** (developed by Google) uses a **static computation graph**, meaning the model architecture is defined before execution. This approach offers performance benefits, especially in production environments. TensorFlow integrates well with:
  - **TensorBoard** (for visualization)
  - **TensorFlow Lite** (for mobile deployment)
  - **TensorFlow Serving** (for model deployment)
- **PyTorch** (developed by Facebook AI Research) uses a **dynamic computation graph**, which is built on-the-fly during execution. This makes PyTorch highly flexible and intuitive, particularly useful in research where models change frequently.

When to choose:

- ✅ Use **TensorFlow** for production environments, robust deployment, or mobile/edge devices.
- ✅ Use **PyTorch** for research, experimentation, and ease of debugging with Pythonic code.

Summary:
PyTorch is often preferred in **academia and research**, while TensorFlow is widely adopted in **industry and production** settings.

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

Answer:
Jupyter Notebooks are essential in AI and data science as they combine code, output, visualization, and documentation in a single document.

1. **Exploratory Data Analysis (EDA):**
   Before training models, Jupyter allows step-by-step exploration of datasets using Python and visualization libraries like Matplotlib and Seaborn. This helps detect patterns, outliers, or class distributions, and allows documentation using markdown cells.

2. **Rapid Prototyping and Experimentation:**
   Developers can test small code blocks incrementally, adjust hyperparameters, evaluate models, and visualize progress without leaving the notebook. This makes it ideal for quick iteration and collaborative sharing.

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

Answer:
spaCy is a modern NLP library optimized for industrial use. Unlike basic string methods like `.split()`, `.find()`, or `.replace()`, spaCy brings linguistic intelligence to text processing.

Key Enhancements:

- **Context-Aware Tokenization:**
  spaCy handles punctuation, contractions, and special entities correctly (e.g., treating "U.S.A." as one token).

- **Part-of-Speech Tagging and Dependency Parsing:**
  spaCy identifies grammatical roles (noun, verb, adjective, etc.) and syntactic relationships between words.

- **Named Entity Recognition (NER):**
  spaCy can extract key entities from text such as:

  - “Apple” → *Organization*
  - “Beats” → *Organization*
  - “2014” → *Date*
  - “$3 billion” → *Money*

These features make spaCy ideal for chatbots, document classification, and information extraction — far beyond what simple string manipulation can achieve.

---

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Aspect	Scikit-learn	TensorFlow
Target Applications	Best for classical ML tasks (e.g., regression, decision trees, SVM, clustering).	Best for deep learning tasks (e.g., CNNs, RNNs, large-scale neural networks).
Ease of Use	Very beginner-friendly with a consistent API (`.fit()` , `.predict()` , `.score()`).	Slightly complex; uses computational graphs. TensorFlow 2.x is more user-friendly with Keras.
Community Support	Mature and strong community; widely used in academia and traditional ML projects.	Very active community in deep learning, backed by Google with rich documentation and tools.

Summary:

- Use **Scikit-learn** for traditional ML and when starting out.
- Use **TensorFlow** for developing deep learning models, large-scale training, and production deployments.

🧠 Part 3: Ethics & Optimization

This section explores ethical considerations in AI models and demonstrates how to troubleshoot a faulty TensorFlow model.

✅ 1. Ethical Considerations

⚠️ A. Potential Biases in Your Models

a) **MNIST (Digit Classifier)**

- **Bias Type**: Model bias due to **overrepresentation of certain digits** or **imbalanced data augmentation**.
- **Impact**: The model may perform better on digits like “1” and worse on ambiguous ones like “5” or “9”.
- **Example**: A digit drawn sloppily or from a different cultural handwriting style (e.g., Arabic, left-handed) may be misclassified.

b) **Amazon Reviews (NER + Sentiment)**

- **Bias Type**: **Lexical bias** — based on limited positive/negative word lists.
- **Impact**: Fails to detect sarcasm, cultural nuance, or contextual meaning.
- **Example**: “This phone is *not bad at all*” would score as **negative**, though it’s actually **positive**.

In [ ]:

✅ B. Mitigation Using AI Fairness Tools

✅ TensorFlow Fairness Indicators

- A tool to evaluate **fairness metrics** across different slices of data (e.g., age, gender, etc.)
- You could apply it in MNIST if metadata (e.g., country, school, or writing hand) were available.

In [3]:
```
# Not required for this task, but here's an example usage idea:
# Evaluate performance disparity on digits drawn by left-handed vs. right-handed in
```

✅ spaCy Rule-Based Mitigation

- You can improve fairness and reduce sentiment misclassification by:
  - Expanding the sentiment word lists based on **domain-specific** vocabulary.
  - Using **dependency parsing** to detect negations.
  - Integrating with libraries like **TextBlob** or **VADER** for better handling of nuanced sentiment.

In [4]:
```
# Example tweak to reduce bias in spaCy rule-based sentiment:
if "not good" in review.lower():
    neg += 1
    pos -= 1  # override naive match
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 # Example tweak to reduce bias in spaCy rule-based sentiment:
----> 2 if "not good" in review.lower():
      3     neg += 1
      4     pos -= 1  # override naive match

NameError: name 'review' is not defined
```

✅ 2. Troubleshooting Challenge

Let's simulate a buggy TensorFlow script and show how to fix it.

❌ Buggy Code Snippet (Hypothetical)

In [ ]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(128, input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
model.fit(x_train, y_train, epochs=5)
```

❗ What's Wrong?

Issue	Description
🟥 Input Shape	`Dense` expects a 1D input, not a 2D image `(28,28)`
🟥 Loss Function	Using `categorical_crossentropy` but labels likely are not one-hot encoded
🟥 Missing Flatten Layer	Need to flatten 2D input before Dense

✅ Fixed Version

In [ ]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),  # Fix input shape
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

# If using sparse labels (integers), use sparse_categorical_crossentropy
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a

model.fit(x_train, y_train, epochs=5)
```