Assignment Report: Mastering the AI Toolkit

Members: Clarence Mabeya

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?

The primary difference historically was their graph-building approach.

- **TensorFlow (1.x):** Used "Define-and-Run" (static graphs). You would first define the entire computation graph, then execute it in a session. This was great for optimization and deployment but harder to debug.
- **PyTorch:** Uses "Define-by-Run" (dynamic graphs). The graph is built on-the-fly as the code executes. This feels more "Pythonic," is easier to debug, and is more flexible for dynamic models (like RNNs with variable inputs).

Today, with **TensorFlow 2.x and Eager Execution** as the default, both frameworks are very similar and use dynamic graphs. The key differences are now more about their ecosystems:

- **TensorFlow:** Has a more mature and extensive production ecosystem, including **TFX (TensorFlow Extended)** for end-to-end pipelines, **TF Lite** for on-device/mobile deployment, and **TensorFlow.js** for running models in the browser. Its high-level API, **Keras**, is also exceptionally user-friendly for beginners.
- **PyTorch:** Is often favored in the research community for its flexibility and clean API. Its path to production is rapidly maturing with tools like **TorchServe** and **TorchScript**.

When to choose:

- **Choose TensorFlow if:** Your primary goal is production deployment, scalability, or deploying to mobile/web environments.

- **Choose PyTorch if:** You are in a research-heavy environment, need maximum flexibility for complex or dynamic model architectures, or prefer a more "Pythonic" coding style.

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

1. **Exploratory Data Analysis (EDA):** Jupyter Notebooks are ideal for EDA because they allow you to mix executable code, data visualizations (like plots from Matplotlib or Seaborn), and narrative text (Markdown) in one document. A developer can load a dataset, check for missing values, plot distributions, and write down their observations cell by cell, creating a clear, shareable record of the data exploration process.
2. **Model Prototyping:** Notebooks provide an interactive environment perfect for rapidly building and testing models. You can define a model architecture in one cell, train it in the next, and immediately evaluate its performance and visualize results in the cells below. This tight feedback loop is much faster for iteration than running a full script repeatedly.

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

Basic string operations (like .split(), .find(), or regular expressions) are "dumb." They only operate on character patterns without any understanding of language, context, or meaning.

spaCy provides a **pre-trained, context-aware pipeline**. When you process text with spaCy, it doesn't just split words; it performs:

- **Tokenization:** Intelligently splits text into words, punctuation, etc. (e.g., handles "don't" as "do" and "n't").
- **Part-of-Speech (PoS) Tagging:** Identifies each word as a noun, verb, adjective, etc.
- **Dependency Parsing:** Understands the grammatical structure of the sentence (e.g., knows the subject and object of a verb).
- **Lemmatization:** Reduces words to their root form ("running" -> "run").
- **Named Entity Recognition (NER):** Identifies real-world entities like "Google" (ORG), "New York" (GPE), or "Steve Jobs" (PERSON).

This linguistic understanding allows you to perform complex tasks easily. For example, instead of guessing if "Apple" is a fruit or a company, spaCy uses the context to label it as an ORG.

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Criterion	Scikit-learn (sklearn)	TensorFlow (TF)
Target Applications	**Classical Machine Learning.** Ideal for structured (tabular) data. Includes algorithms for regression, classification (SVM, Decision Trees, Random Forests), clustering (k-Means), and dimensionality reduction (PCA).	**Deep Learning (Neural Networks).** Ideal for unstructured data (images, text, audio). Used to build CNNs, RNNs, Transformers, etc. Can also do classical ML, but it's not its primary strength.
Ease of Use for Beginners	**Extremely high.** The API is famously consistent: model.fit(), model.predict(), model.transform(). It's the standard starting point for anyone learning ML.	**High (with Keras).** The Keras API within TensorFlow makes building complex neural networks very simple. However, the underlying *concepts* of deep learning (layers, loss functions, optimizers) are more abstract than sklearn's.
Community Support	**Massive and mature.** As the cornerstone of data science in Python for over a decade, it has extensive documentation, tutorials, and countless examples on sites like Stack Overflow.	**Massive and growing.** Backed by Google, it has a huge community, dedicated conferences, and extensive resources. Community support is very strong for both deep learning and production MLOps.

Part 3: Ethics & Optimization

1. Ethical Considerations

MNIST Model:

- **Potential Bias:** The MNIST dataset, while standard, is not perfectly representative of all handwriting. It may be biased towards a specific style of clear, well-formed digits.
- **Impact:** The model might have lower accuracy for individuals with different handwriting styles, such as those from different cultural backgrounds, the elderly, or people with motor disabilities, leading to exclusion or errors in a real-world application (e.g., a check-reading system).

Amazon Reviews Model:

- **Potential Bias (NER):** A pre-trained NER model (like en_core_web_sm) is trained on general web text. It may fail to recognize new, niche, or non-English brand/product names, or it might misclassify them.
- **Potential Bias (Sentiment):** Sentiment is highly contextual.
    - **Domain Bias:** A model might interpret "This speaker is sick!" as negative, when in slang, it's positive.
    - **Demographic Bias:** The model may be biased by the writing styles of the dominant demographic in the training data, potentially misunderstanding or misinterpreting reviews from other groups.

Mitigation:

- **TensorFlow Fairness Indicators (TFFI):** For the MNIST model, if we had demographic data (which we don't), TFFI would be perfect. We could slice the dataset by "writing style" (if labeled) and use TFFI to measure and compare accuracy across these slices, identifying where the model is failing.
- **spaCy's Rule-Based Systems:** This is ideal for mitigating the Amazon review biases.
    - **NER:** We can use spaCy's EntityRuler to add a list of custom product/brand names that the model misses. This rule-based list can augment or override the model's predictions.
    - **Sentiment:** We can use the rule-based approach (like spacytextblob or our own rules) to correct known sentiment errors. For example, we can add a rule that "sick" in the context of "speaker" or "headphone" is positive, overriding the default negative polarity.

2. Troubleshooting Challenge

Provided "Buggy" Code:

```
# --- BUGGY CODE ---
# import tensorflow as tf
# from tensorflow.keras import layers, models
#
# (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# x_train, x_test = x_train / 255.0, x_test / 255.0
#
# model = models.Sequential([
#     layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28)), # Bug 1
#     layers.MaxPooling2D((2, 2)),
#     layers.Dense(128, activation='relu'), # Bug 2
#     layers.Dense(10, activation='softmax')
# ])
#
# model.compile(optimizer='adam',
#               loss='binary_crossentropy', # Bug 3
#               metrics=['accuracy'])
#
# model.fit(x_train, y_train, epochs=5)
# model.evaluate(x_test, y_test)
```

Fixed and Debugged Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np

# Load and normalize data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# --- FIX 1: Add channel dimension for Conv2D ---
# Conv2D layers expect a "channel" dimension (1 for grayscale).
# We reshape (60000, 28, 28) to (60000, 28, 28, 1)
x_train = x_train[..., np.newaxis]
x_test = x_test[..., np.newaxis]

model = models.Sequential([
    # --- FIX 1 (cont.): Update input_shape to (28, 28, 1) ---
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
```

```
layers.MaxPooling2D((2, 2)),

    # --- FIX 2: Add Flatten layer ---
    # We must flatten the 2D feature map from the Conv/Pool layers
    # before passing it to a 1D (Dense) layer.
    layers.Flatten(),

    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax') # Output layer (10 classes)
])

# --- FIX 3: Change the loss function ---
# 'binary_crossentropy' is for 2-class (binary) problems.
# 'categorical_crossentropy' is for multi-class problems (0-9)
# We use 'sparse_categorical_crossentropy' because our labels (y_train)
# are integers (0, 1, 2...) and not one-hot encoded ([0,1,0,0...]).
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

print("--- Model Summary ---")
model.summary()

print("\n--- Training Model ---")
model.fit(x_train, y_train, epochs=5, validation_split=0.1)

print("\n--- Evaluating Model ---")
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc*100:.2f}%")
```

Explanation of Bugs Fixed:

1. **Dimension Mismatch (Input):** Conv2D layers in TensorFlow expect a 4D input: (batch_size, height, width, channels). The original MNIST data is 3D: (batch_size, height, width).
    a. **Fix:** We added a channel dimension using np.newaxis (e.g., x_train[..., np.newaxis]) and updated the input_shape in the first layer to (28, 28, 1).
2. **Dimension Mismatch (Flatten):** The Conv2D and MaxPooling2D layers output a 3D tensor (feature map). This cannot be fed directly into a Dense (fully connected) layer, which expects a 1D vector.

    a. **Fix:** We added a layers.Flatten() layer to unroll the 3D feature map into a 1D vector before it reaches the Dense layer.
3. **Incorrect Loss Function:** The problem is a multi-class classification (10 digits), but the loss function was binary_crossentropy, which is for binary (2-class) problems.
    a. **Fix:** We changed the loss to sparse_categorical_crossentropy. This is the correct choice for multi-class classification where the labels are provided as integers (like 0, 1, 2...9) rather than one-hot encoded vectors.