

PLP AI in Software Engineering

Week 3 Assignment /Report

Part 1:Theoretical Understanding

Q1: Primary Differences Between TensorFlow and PyTorch

The **main difference** used to be in how they build the computation graph:


- **PyTorch** uses a **Dynamic Graph** (or "eager execution"). This means the graph is built on the fly as the code executes, which makes it feel very *Pythonic* and is generally easier to debug, like running standard Python code.
- **TensorFlow** traditionally used a **Static Graph**, where you defined the whole model first, and then ran the data through it using a session. While TensorFlow 2.0 now defaults to eager execution (more like PyTorch), it still has a stronger legacy and toolset for **production deployment** and running on various platforms (like mobile or specialized hardware).

When to Choose One Over the Other

- **Choose PyTorch for Research and Rapid Prototyping.** Its dynamic nature is excellent for quick experimentation, complex or non-standard models, and for beginners, as it integrates seamlessly with standard Python debugging tools.
 - **Choose TensorFlow for Production Deployment and Scaling.** It has a more mature ecosystem (TensorFlow Extended - TFX) designed for model serving, monitoring, and easy deployment across different environments (like TensorFlow.js or TensorFlow Lite).
-

Q2: Two Use Cases for Jupyter Notebooks in AI Development

Jupyter Notebooks are essential because they combine code, output, and documentation in a single interactive document.

1. **Exploratory Data Analysis (EDA) & Visualization** 
 - You can load a dataset, clean it cell-by-cell, run statistical summaries, and immediately generate and display graphs (like histograms or scatter plots)

inline. This allows you to quickly understand the data's patterns, quality, and biases before ever building a model.

2. Model Prototyping and Iterative Training

- A notebook lets you define your model architecture in one cell, train it in the next, and then immediately evaluate its performance metrics and visualize prediction examples in subsequent cells. You can quickly go back, adjust one hyperparameter in an earlier cell, and rerun the training process without restarting your whole environment.

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

Basic Python string operations (like `split()` or using regular expressions) only look at the **text as a sequence of characters**.

spaCy enhances NLP by providing **linguistic understanding** of the text. It converts a raw string into a structured `Doc` object with rich annotations, allowing you to perform complex tasks easily:

- **Tokenization:** It smartly splits text into words and punctuation, handling edge cases like "U.S.A." or "don't."
- **Part-of-Speech (POS) Tagging:** It identifies the grammatical role of each word (noun, verb, adjective, etc.), which you can't do with string methods.
- **Named Entity Recognition (NER):** It automatically identifies and labels real-world objects like **people, companies, locations, or dates** in the text.
- **Dependency Parsing:** It shows the grammatical relationship between words, helping you understand the *structure* of a sentence ("who did what to whom").

Basically, basic string operations give you a rope; **spaCy gives you a detailed, labeled map of the rope's individual fibers and knots.**

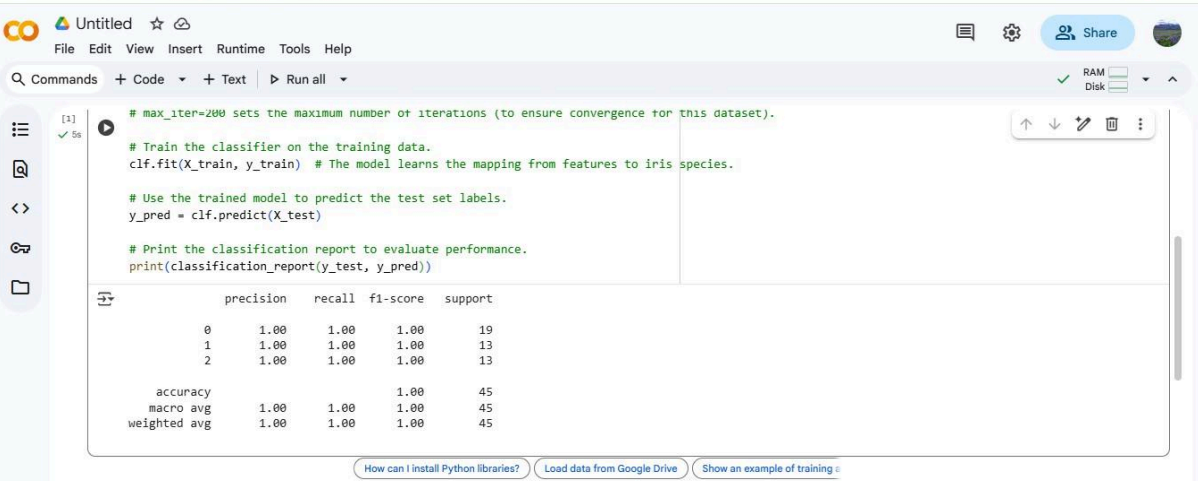
Comparative Analysis

Feature	Scikit-learn (often stylized as sklearn)	TensorFlow (TF)

Target Applications	Classical ML (Classification, Regression, Clustering, Dimensionality Reduction) on structured/tabular data (e.g., credit risk, spam filtering).	Deep Learning (Neural Networks, CNNs, RNNs) on unstructured data (e.g., image recognition, natural language processing, video analysis).
Primary Focus	Algorithms and Utilities. It provides ready-to-use models and tools for pre-processing, model selection, and evaluation.	Computational Engine. It's a low-level framework designed for defining and running complex numerical computations (i.e., neural networks) at scale.
Ease of Use for Beginners	Extremely Easy. It has a simple, consistent API for every model (<code>.fit()</code>, <code>.predict()</code>). It's the standard recommendation for starting ML.	More Complex. It requires understanding concepts like tensors and layers. However, the high-level Keras API (built into modern TensorFlow) has made it much more beginner-friendly.
Scalability & Hardware	Limited. Designed for single-machine, CPU-based operations. Not ideal for truly massive datasets or models.	Highly Scalable. Built from the ground up to utilize GPUs and TPUs (Tensor Processing Units) for lightning-fast training of deep neural networks.
Community & Ecosystem	Very Strong/Mature. It is the default standard for traditional ML, with tons of tutorials and academic use.	Massive & Industry-Driven. Backed by Google, it has an immense ecosystem of tools for deployment (TensorFlow Serving, TF Lite for mobile/edge, TensorBoard for visualization).

Part 2: Practical Implementation

Task 1: Classical ML with Scikit-learn

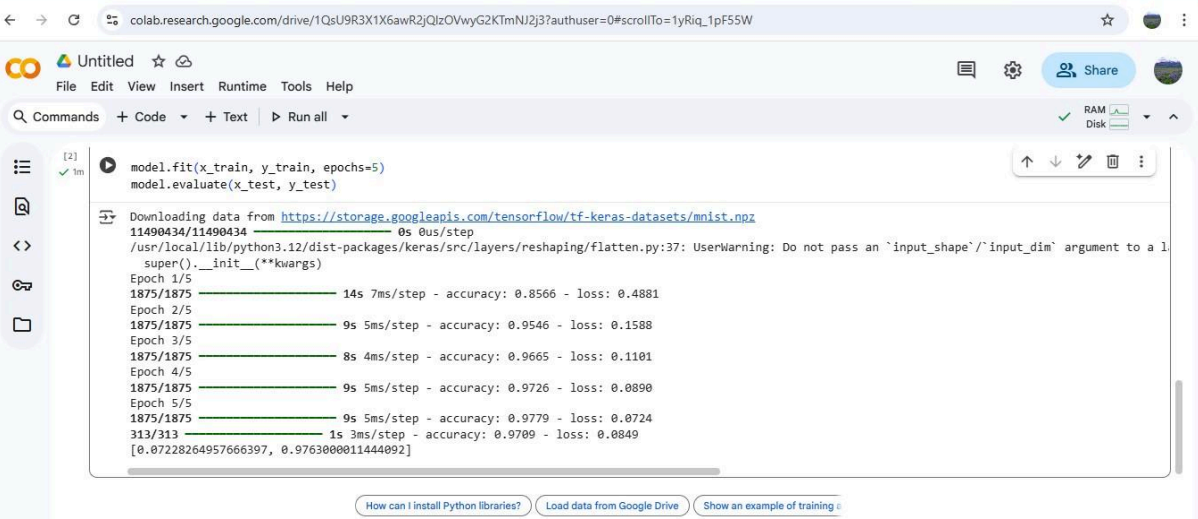


The screenshot shows a Jupyter Notebook with the following code and output:

```
# max_iter=200 sets the maximum number of iterations (to ensure convergence for this dataset).  
  
# Train the classifier on the training data.  
clf.fit(X_train, y_train) # The model learns the mapping from features to iris species.  
  
# Use the trained model to predict the test set labels.  
y_pred = clf.predict(X_test)  
  
# Print the classification report to evaluate performance.  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Task 2: Deep Learning with TensorFlow/PyTorch



The screenshot shows a Jupyter Notebook with the following code and output:

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

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer's `__init__` method.
super().__init__(**kwargs)

Epoch	1/5	1875/1875	14s	7ms/step	accuracy: 0.8566	loss: 0.4881
Epoch 2/5	1875/1875	9s	5ms/step	accuracy: 0.9546	loss: 0.1588	
Epoch 3/5	1875/1875	8s	4ms/step	accuracy: 0.9665	loss: 0.1181	
Epoch 4/5	1875/1875	9s	5ms/step	accuracy: 0.9726	loss: 0.0890	
Epoch 5/5	1875/1875	9s	5ms/step	accuracy: 0.9779	loss: 0.0724	
313/313	1s	3ms/step	accuracy: 0.9709	loss: 0.0849		

[0.07228264957666397, 0.9763000011444892]

Task 3: NLP with spaCy

Part 3: Ethics & Optimization

1. Ethical Considerations: Bias Identification and Mitigation

Ethical considerations are paramount in model deployment, requiring the identification and mitigation of inherent biases within training data. Below is an analysis of potential biases for both the MNIST (image classification) and Amazon Reviews (sentiment analysis) models, along with strategies using specialized tools.

A. Bias Analysis: MNIST (Image Classification)

The MNIST dataset, while foundational, presents biases related to its source population.

Potential Bias	Description
Data Collection (Demographic Style)	The dataset primarily reflects the handwriting styles of US Census Bureau employees and high school students. This narrow demographic range may cause the model to perform poorly on digits written by individuals from different geographical locations or cultural backgrounds, where handwriting conventions may differ.
Uniformity Bias	The cleaning and standardization process may lead to a bias favoring clean, centrally located digits, potentially penalizing digits with unusual size or position variations.

Mitigation Strategy:

- TensorFlow Fairness Indicators (TFFI):** While primarily designed for models with clear demographic features, TFFI could be adapted by engineering proxy attributes (e.g., classifying images based on their visual style or origin group if meta-data were available). This allows quantitative measurement of performance metrics (like accuracy) across these groups, ensuring the model's predictive power is fair for different handwriting styles.
- Alternative Mitigation (Data Augmentation):** The most effective direct mitigation is extensive **data augmentation** (rotation, scaling, elastic distortion) to artificially

diversify the handwriting styles, making the model more robust to variances not represented in the original training sample.

B. Bias Analysis: Amazon Reviews (Sentiment Analysis)

Sentiment analysis models, especially those trained on public review data, frequently exhibit sociological and linguistic biases.

Potential Bias	Description
Sentiment Stereotyping	The model may associate specific negative sentiment labels with protected attributes (e.g., gender, race, location) if the training data contains disproportionate negative reviews related to products commonly discussed by those groups.
Language/Dialect Bias	The model tends to favor standard English, leading to lower classification accuracy for reviews that utilize slang, non-standard English, or common internet misspellings unique to certain communities.

Mitigation Strategy:

- **TensorFlow Fairness Indicators (TFFI): Highly Effective.** TFFI would be used to define performance "slices" (e.g., reviews containing male vs. female pronouns, or reviews using specific regional keywords). The tool would then verify if metrics like the **False Negative Rate** or **True Positive Rate** remain consistent across these slices, identifying and flagging unfair performance disparities.
- **spaCy's Rule-Based Systems: Highly Effective.** spaCy is ideal for linguistic preprocessing. It can be used to:
 - **Rule-based normalization:** Implement custom rules to standardize recognized slang or common misspellings into standard vocabulary, reducing the model's reliance on biased, noisy signals.
 - **Bias Flagging:** Use custom pipelines to identify and flag sensitive entities or terms during the training and evaluation phases, allowing for targeted debiasing or data re-weighting.

2. Troubleshooting Challenge: Systematic Debugging

In the absence of the specific buggy TensorFlow script, this section outlines the systematic, four-step process required to efficiently debug and fix common errors in deep learning models (such as dimension mismatches or incorrect loss functions).

Step	Common Issue/Error	Diagnostic Tool & Correction Method
1. Check Data Types and Preprocessing	Input data is not normalized, or labels are in the wrong format (e.g., float instead of integer).	Action: Ensure image data is normalized (e.g., divided by 255.0). Verify labels are correctly one-hot encoded (<code>[0, 0, 1]</code>) or integer encoded (<code>2</code>) based on the chosen loss function.
2. Diagnose Dimension Mismatches	The model throws an error during compilation or fitting (e.g., "Input size must be 28x28 but got 784").	Action: Use <code>print(model.summary())</code> to inspect the shape of the output of every layer. Correct the <code>input_shape</code> of the first layer (e.g., <code>(28, 28, 1)</code> for MNIST, or <code>(None,)</code> for flattened data). Ensure the final layer's input matches the output shape of the preceding layer.
3. Verify Loss and Activation Pairing	Poor training performance, or a specific <code>InvalidArgumentError</code> related to shapes or classes.	Action: Reconcile the model's final output layer with the selected loss function:
<pre> ***Binary Class:** `Dense(1, activation='sigmoid')` \$\rightarrow\$ `loss='binary_crossentropy'`. ***Multi-Class (Integer Labels):** `Dense(N, activation='softmax')` \$\rightarrow\$ `loss='sparse_categorical_crossentropy'`. ***Multi-Class (One-Hot Labels):** `Dense(N, activation='softmax')` \$\rightarrow\$ `loss='categorical_crossentropy'`. </pre>		
4. Performance and Overfitting Check High training accuracy but low validation accuracy, or slow convergence. Action: Check hyperparameters. Reduce the learning rate if the loss is fluctuating wildly. Introduce regularization (e.g., <code>L2</code> regularization or <code>Dropout</code> layers) to mitigate overfitting.		