WEEK 3 ASSIGNMENT

Part 1: Theoretical Understanding (40%)

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 two of the most popular deep learning frameworks. They both support building and training complex neural networks, but they differ in philosophy, usability, and ecosystem. The primary differences are;.

a. Programming Style and Syntax

TensorFlow uses static computation graphs, the code is less intuitive. TensorFlow 2.x added eager execution by default, similar to PyTorch. PyTorch uses dynamic graphs (define-by-run), is similar to python, easier to read, write, and debug. It is favoured in research due to its flexibility.

b. Ecosystem and Deployment

TensorFlow has a more diverse ecosystem with strong support for production and deployment e.g., TensorFlow Serving, TensorFlow Lite, TensorFlow.js. There are tools like TensorBoard for visualization and TFLite for mobile/embedded AI. PyTorch was originally more research - focused, but deployment has improved with TorchServe and ONNX. It is still catching up in mobile/web deployment compared to TensorFlow.

c. Community and Adoption

PyTorch is popular in academia and research and used by Facebook and many top AI researchers. TensorFlow is backed by Google, used in many industry applications. Larger overall ecosystem due to its longer presence and tools. pyTorch is good for rapid prototyping and research while TensorFlow is better for industry scale ML workflows.

d. Tools and Extensions

TensorFlow includes Keras as a high-level API and has dedicated tools for various tasks e.g. TF Agents for RL, TF Hub for pre-trained models. PyTorch has rich third-party libraries such as Hugging Face Transformers, fast ai. It also includes torch.nn, torchvision, and PyTorch Lightning for more structure.

Which to Choose?

- For learning and research: PyTorch
- For production deployment, mobile/web apps: TensorFlow
- If you want strong community support for NLP/transformers: PyTorch + Hugging Face
- If you need enterprise-level tools and ecosystem: TensorFlow

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

Two use cases for Jupyter Notebooks in AI development:

a. Exploratory Data Analysis (EDA) and Visualization

Before building AI models, data scientists use Jupyter Notebooks to explore, clean, and visualize datasets. Notebooks support step-by-step execution with rich outputs (e.g., graphs, tables), making it easy to inspect data and detect issues like missing values, outliers, or class imbalance. An example is using `pandas` and `matplotlib` to analyze customer churn data and create visualizations showing customer trends before model training.

b. 2. Model Development and Experimentation

Building, training, and testing machine learning or deep learning models interactively. You can tweak hyperparameters, visualize training progress, and debug issues in real time—all within one environment. An example is training a neural network using TensorFlow or PyTorch, tracking accuracy/loss with plots, and experimenting with different architectures or optimizers.

Jupyter Notebooks are ideal for rapid prototyping, documentation, and collaboration, especially in AI workflows where experimentation and visual feedback are essential.

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

SpaCy provides advanced, efficient, and accurate natural language processing capabilities that go far beyond basic Python string operations. It does this by;

a. **Linguistic Understanding** - spaCy uses trained statistical models to understand parts of speech, sentence structure, named entities, etc. Basic Python only offers simple string manipulation (e.g., `split()`, `find()`, `replace()`), with no understanding of grammar or context. spaCy can identify that "Apple" is a company in “Apple released a new iPhone,” while basic string ops just see it as text.
b. **Tokenization and Lemmatization** -   Accurately splits text into tokens (words, punctuation, etc.), handles lemmatization (reducing words to their base form), and understands context. Basic Python can split strings using `.split()` but struggles with complex sentence structures or punctuation. *eg* spaCy tokenizes contractions like “don’t” correctly into "do" and "not"; basic string methods cannot do this.
c. **Named Entity Recognition (NER)** - spaCy: Automatically detects entities like names, locations, dates, and organizations. Basic Python: Requires manual pattern matching (e.g., with regex), which is error-prone and hard to scale.

d. **Efficiency and Speed** - spaCy is built for speed with optimized Cython code, making it suitable for production-level NLP pipelines while basic Python is not optimized for large-scale or real-time text processing.

spaCy turns raw text into structured, machine-understandable data, making it far more powerful and practical for real-world NLP applications than basic Python string methods.

2. Comparative Analysis

Compare Scikit-learn and TensorFlow in terms of:

- ***Target applications (e.g., classical ML vs. deep learning).***
- ***Ease of use for beginners.***
- ***Community support.***

Target applications

The primary focus for SciKit-learn is classical Machine Language while TensorFlow focus is on deep learning and Neural networks. In terms of algorithms, SciKit-learn uses SVM, Decision Trees, Random Forests, KNN,etc and is mainly used for Fraud detection, customer segmentation and linear regression while TensorFlow uses Neural networks such as CNNs, RNNs, Transformers, etc. and is used for Image recognition, speech-to-text, NLP with transformers.

Use Scikit-learn when your problem is best solved with classical ML algorithms; Use TensorFlow when you're working with deep learning or large-scale unstructured data (e.g., images, text).

Ease of use for beginners.

The learning curve for Scikit-learn is low as its a simple API, easy to pick up for beginners; TensorFlow has a moderate to high learning curve due to more complex APIs. The code for SciKit-learn is simpler, readable and more pythonic while for TensorFlow its more complex. This makes SciKit-learn easier to learn for first-time learners in ML. TensorFlow is suited for users with basic ML understanding looking to learn deep learning.

Community support.

SciKit-learn is widely used in education and classical ML tasks, while TensorFlow is extremely popular, especially for deep learning. SciKit-learn has lots of tutorials and examples but limited deployment tools. Tensor Flow has a massive ecosystem, including Keras, TF Hub, TF Lite. SciKit-learn is for common in data science and analytics roles and TensorFlow for common in AI research, production, and big tech.

Both have strong communities, but TensorFlow has a larger and more diverse ecosystem, especially for production and deployment. It is advisable to start with Scikit-learn if you are new to ML or working on structured data and traditional models and move to TensorFlow when you need to build powerful deep learning models or deploy them at scale.

Part 2: Practical Implementation (50%)

Task 1: Classical ML with Scikit-learn -

- ***Dataset***: *Iris Species Dataset*
- ***Goal***:
    1. *Preprocess the data (handle missing values, encode labels).*
    2. *Train a **decision tree classifier** to predict iris species.*
    3. *Evaluate using accuracy, precision, and recall.*
- ***Deliverable***: *Python script/Jupyter notebook with comments explaining each step.*

Response: ***week3AI4SEassignment/iris.ipynb at main · mueni254/week3AI4SEassignment***

Task 2: Deep Learning with TensorFlow/PyTorch

- ***Dataset***: *MNIST Handwritten Digits*
- ***Goal***:
    1. *Build a **CNN model** to classify handwritten digits.*
    2. *Achieve >95% test accuracy.*
    3. *Visualize the model's predictions on 5 sample images.*
- ***Deliverable***: *Code with model architecture, training loop, and evaluation.*

Response:
https://colab.research.google.com/drive/1w_sEuChN1N56MWX5Ia4NKWvqKcGUUkOA?usp=sharing

Task 3: NLP with spaCy

- ***Text Data**: User reviews from Amazon Product Reviews.*
- ***Goal**:*
    1. *Perform **named entity recognition (NER)** to extract product names and brands.*
    2. *Analyze sentiment (positive/negative) using a rule-based approach.*
- ***Deliverable**: Code snippet and output showing extracted entities and sentiment.*

Response
https://colab.research.google.com/drive/17rgpP1zTEQ7jBympOfGwBTvU_bahxfPx?usp=sharing

Part 3: Ethics & Optimization (10%)

1. Ethical Considerations

Identify potential biases in your MNIST or Amazon Reviews model. How could tools like TensorFlow Fairness Indicators or spaCy's rule-based systems mitigate these biases?

Potential Biases in the Amazon Reviews Model

1. Training Data Bias (Reviews) - Amazon reviews often reflect:

- Cultural/language bias (e.g., U.S.-centric expressions)
- Reviewer demographics (e.g., gender, age)
- Brand popularity bias (e.g., popular brands get more reviews)

Hence Sentiment and NER might behave differently depending on who wrote the review or what product is being reviewed.

2. VADER Lexicon Bias

- VADER’s dictionary reflects **social media-style sentiment**, not necessarily formal or domain-specific language.
- Words like “killer” or “insane” might be positive in some contexts (“killer deal”) but misinterpreted.

Sentiment scores may not reflect the true tone in niche domains (e.g., tech reviews, healthcare products).

3. spaCy NER Bias

- spaCy’s models are trained on **general-purpose text** (like news articles).
- Uncommon product names or brands may not be recognized at all.
- Models may also over-identify familiar Western brands, but miss local or less-known entities.

Some brands or products may be underrepresented or missed entirely.

How spaCy's Rule-Based Tools Can Help Mitigate These Biases

spaCy offers **rule-based components** (like `Matcher`, `PhraseMatcher`, and `EntityRuler`) that you can use to **augment or override** the statistical model.

1. EntityRuler for Custom Brands/Products

You can explicitly define product and brand names regardless of how frequent or "popular" they are from spacy.pipeline import EntityRuler. This ensures even obscure brands are recognized.

2. PhraseMatcher for Bias-Free Extraction

Rather than rely on a model's interpretation of context, PhraseMatcher lets you extract exact terms. Consistent and transparent entity recognition.

3. Transparent Sentiment Rules (VADER)

With VADER, you can:

- Inspect the lexicon
- Customize sentiment scores for specific words
- Add domain-specific sentiment rules

This gives you full control over how sentiment is scored — ideal for avoiding surprise behavior.

In Summary

Bias Source	Mitigation via spaCy Rule-Based Tools
Data bias (reviewers)	Hard to avoid, but transparent logic helps
NER brand/product gaps	Use `EntityRuler` or `PhraseMatcher`
Sentiment lexicon bias	Customize VADER lexicon or add overrides
Model generalization bias	Override with consistent, rule-based logic

2. Troubleshooting Challenge

Buggy Code: A provided TensorFlow script has errors (e.g., dimension mismatches, incorrect loss functions). Debug and fix the code.

*1. **Dimension Mismatches***

Problem: Your input data shape doesn't match the model's expected input shape.

Fix:
Confirm input shape when building the model, e.g. `(28, 28, 1)` for grayscale images.
Ensure data tensors have batch dimension, e.g. `(batch_size, 28, 28, 1)`.
Use `.reshape` or `[..., np.newaxis]` to fix missing channel dims.

*2. **Incorrect Loss Function***

Problem: Using a wrong loss for classification, e.g., `mse` instead of `sparse_categorical_crossentropy` for integer labels.

Fix:

For multi-class classification with integer labels, use: loss='sparse_categorical_crossentropy'

If your labels are one-hot encoded vectors, use: loss='categorical_crossentropy'

3. Activation and Output Layer Mismatch

Problem: Output layer size or activation doesn't match number of classes.

Fix:
For MNIST (10 digits), last layer must be: layers.Dense(10, activation='softmax')

If binary classification, output layer usually has 1 neuron with sigmoid:
layers.Dense(1, activation='sigmoid')

4. Model Compilation Issues

Ensure optimizer and metrics are properly set, for example:

```
model.compile(optimizer='adam',

              loss='sparse_categorical_crossentropy',

              metrics=['accuracy'])
```

5. Batch Dimension Issues During Prediction

When predicting a single image, add batch dimension:

```
single_image = X_test[0]  # shape (28, 28, 1)

single_image = np.expand_dims(single_image, axis=0)  # shape (1, 28, 28, 1)

prediction = model.predict(single_image)
```

6. Check TensorFlow Version Compatibility

Some syntax or APIs change between TF 1.x and TF 2.x. Confirm you are using TensorFlow 2.x, preferably 2.8+.

Bonus Task (Extra 10%)

- ***Deploy Your Model:** Use Streamlit or Flask to create a web interface for your MNIST classifier. Submit a screenshot and a live demo link.*