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?

Primary Differences:

- **Computation Graph Paradigm:**
  - **TensorFlow (TF 1.x): Static Graphs (Define-and-Run):** You first define the entire computation graph (the neural network structure) and then execute it. This allows for compiler optimizations and easier deployment but can make debugging harder.
  - **PyTorch: Dynamic Graphs (Define-by-Run):** The graph is built on the fly as operations are performed. This makes PyTorch more flexible and intuitive, especially for debugging (as you can inspect values at any point) and for models with dynamic structures (like some RNNs). TensorFlow 2.x, with Keras and Eager Execution, has largely adopted a dynamic graph philosophy, making it much more similar to PyTorch in this regard.

- **Debugging:**
  - **PyTorch:** Generally considered easier to debug due to its dynamic graphs, allowing standard Python debugging tools to be used directly.

  - **TensorFlow (TF 1.x):** Debugging static graphs could be more challenging, often requiring specialized tools like `tf.Session` and `tf.debugger`. TF 2.x's Eager Execution significantly improves debugging by behaving like standard Python.

- **Deployment & Production Readiness:**
  - **TensorFlow:** Historically, TensorFlow has had a stronger ecosystem for production deployment (e.g., TensorFlow Serving for production-grade model serving, TensorFlow Lite for mobile/edge devices, TensorFlow.js for web).
  - **PyTorch:** While rapidly catching up (e.g., TorchServe), its production ecosystem has traditionally been less mature than TensorFlow's, though it's now widely used in production environments.

- **Learning Curve & Pythonic Nature:**
  - **PyTorch:** Often perceived as more "Pythonic" and intuitive for Python developers, making the learning curve slightly gentler for those already familiar with Python.
  - **TensorFlow (TF 2.x with Keras):** Has greatly simplified its API and made it more user-friendly, reducing the learning curve significantly compared to TF 1.x.

When to Choose One over the Other:

- **Choose TensorFlow when:**
  - **Production Deployment is a High Priority:** If your primary goal is to deploy the model to various environments (mobile, web, large-scale servers) with robust serving capabilities and specialized tools.
  - **Large-Scale Operations/Distributed Training:** Its mature tools for distributed training and its ecosystem are very powerful for massive datasets and models.
  - **Enterprise-Level Projects:** Many larger organizations have standardized on TensorFlow due to its stability, long-term support, and comprehensive ecosystem.

- **Choose PyTorch when:**
  - **Research and Rapid Prototyping:** Its dynamic nature and Pythonic interface make it excellent for experimentation, quickly iterating on ideas, and dealing with complex, custom model architectures.
  - **Academic Environments:** Widely adopted in academia due to its flexibility and ease of use for new research.
  - **Debugging Flexibility:** When you anticipate needing to frequently step through your model's execution and inspect intermediate values.
  - **Pythonic Simplicity:** If you prefer a framework that feels more like traditional Python programming.

In practice, with TensorFlow 2.x, the choice often comes down to personal preference, existing team expertise, and specific deployment needs, as both frameworks are incredibly powerful and converge on many best practices.

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

Jupyter Notebooks are widely adopted in AI development due to their interactive nature and ability to combine code, output, and narrative in a single document. Here are two primary use cases:

1. **Exploratory Data Analysis (EDA) and Prototyping:**

   - **Description:** Jupyter Notebooks provide an ideal environment for the initial stages of an AI project, particularly for understanding and preparing data. Data scientists can load datasets, inspect their structure (e.g., using `df.head()`, `df.info()`), visualize distributions (with libraries like Matplotlib or Seaborn), identify missing values, and quickly experiment with different data preprocessing techniques (e.g., scaling, encoding categorical variables). The cell-by-cell execution allows for immediate feedback on each step, making it easy to iterate and refine data preparation pipelines. For prototyping, developers can rapidly build and test small model architectures or algorithm snippets without the overhead of creating full scripts, seeing the results instantly.
   - **Why it's useful:** This interactive feedback loop significantly accelerates the understanding of data characteristics and the initial design of AI solutions. It helps in identifying potential issues early on and quickly validating hypotheses about data behavior or model performance.

2. **Model Training, Evaluation, and Documenting Workflows:**

   - **Description:** Beyond initial exploration, Jupyter Notebooks are frequently used for the full lifecycle of model training and evaluation, especially for smaller to medium-sized experiments. Developers can write code to define model architectures (e.g., Keras models), set up training loops, and monitor performance metrics (like loss and accuracy) as training progresses. Crucially, the outputs (graphs of training history, confusion matrices, classification reports) are embedded directly within the notebook. Furthermore, Markdown cells can be interspersed with code to add detailed explanations, document design choices, explain evaluation results, or even jot down conclusions and next steps. This creates a living document that serves as both executable code and comprehensive project documentation.
   - **Why it's useful:** The ability to combine code, its output, and rich text explanations in one file makes notebooks excellent for reproducibility, sharing

work with teammates (especially for peer review, as required in this assignment), and presenting findings to non-technical stakeholders. It streamlines the process of demonstrating how data flows through a model and what results are achieved.

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

Basic Python string operations (like `str.split()`, `str.lower()`, `str.replace()`, and regular expressions using the `re` module) are fundamental for manipulating text data. However, they operate at a superficial, character or word-level and lack any true linguistic understanding. spaCy, on the other hand, is a powerful and efficient library specifically designed for advanced Natural Language Processing (NLP), offering a rich set of linguistic features that significantly enhance NLP tasks beyond what basic string operations can achieve.

Here's how spaCy provides significant enhancements:

1. **Intelligent Tokenization:**
    - **Basic String Ops:** `text.split()` might split "don't" into "don" and "t", or separate punctuation incorrectly (e.g., "word." into "word" and ".").
    - **spaCy:** Uses a sophisticated, rule-based tokenizer that understands linguistic nuances. It correctly handles contractions ("don't" becomes "do", "n't"), identifies punctuation as separate tokens, and manages special cases like URLs, emails, and hashtags, providing more accurate and meaningful word units.
2. **Part-of-Speech (POS) Tagging:**
    - **Basic String Ops:** No inherent capability to identify if a word is a noun, verb, adjective, etc.
    - **spaCy:** Assigns grammatical categories (e.g., `NOUN`, `VERB`, `ADJ`) to each token. This is crucial for understanding sentence structure, disambiguating word meanings (e.g., "bank" as a noun vs. a verb), and for subsequent NLP tasks.
3. **Named Entity Recognition (NER):**
    - **Basic String Ops:** Identifying entities like "Apple Inc." as an organization, "Tim Cook" as a person, or "London" as a location would require extensive, brittle, and manually curated regex patterns or lookup tables, which are highly prone to error and difficult to maintain.
    - **spaCy:** Comes with pre-trained statistical models capable of identifying and classifying named entities in text (e.g., `ORG`, `PERSON`, `GPE` for geopolitical entity, `DATE`, `MONEY`). This allows for automatic extraction of crucial

information from unstructured text, which is nearly impossible with basic string operations alone.

4. **Dependency Parsing:**
    - **Basic String Ops:** No understanding of the grammatical relationships between words in a sentence.
    - **spaCy:** Analyzes the grammatical structure of sentences, showing which words modify or relate to others (e.g., identifying the subject, object, and verbs). This is fundamental for advanced tasks like information extraction and semantic understanding.

5. **Lemmatization:**
    - **Basic String Ops:** You might manually

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Here's a comparison between Scikit-learn and TensorFlow across key aspects:

Feature	Scikit-learn	TensorFlow
Target Applications	Primarily **classical machine learning (ML)** algorithms.	Primarily **deep learning (DL)** and neural networks.
	- **Supervised Learning:** Classification (e.g., Logistic Regression, SVMs, Decision Trees, Random Forests), Regression (e.g., Linear Regression, Ridge, Lasso).	- **Deep Neural Networks:** Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Transformers.
	- **Unsupervised Learning:** Clustering (e.g., K-Means, DBSCAN), Dimensionality Reduction (e.g., PCA, t-SNE).	- **Unstructured Data:** Excellent for images, audio, video, complex text data.
	- **Model Selection & Preprocessing:** Cross-validation, hyperparameter tuning, feature scaling, encoding.	- **Large-Scale Training:** Designed for GPU/TPU acceleration and distributed training.
	- Best suited for **structured, tabular data**.	- **Generative Models:** GANs, VAEs.
		- **Reinforcement Learning:** (though often with

Feature	Scikit-learn	TensorFlow
		dedicated libraries built on TF).
Ease of Use for Beginners	**Generally easier and more beginner-friendly.**	**Can be more challenging, but TensorFlow 2.x with Keras has simplified it significantly.**
	- Consistent and intuitive API (`.fit()`, `.predict()`, `.transform()`) across various algorithms.	- Requires understanding of neural network components (layers, activation functions, optimizers, loss functions).
	- Less boilerplate code needed to get a basic model running.	- More verbose for custom architectures.
	- Abstraction of low-level mathematical operations.	- Higher learning curve to fully utilize advanced features (e.g., custom layers, distributed strategies).
Community Support	**Very large, active, and mature community.**	**Massive, global, and highly active community (backed by Google).**
	- Extensive documentation, numerous tutorials, and a strong presence in general data science discussions.	- Abundant official documentation, tutorials, research papers, and a vibrant community of researchers and practitioners.
	- Mature library, so many common problems have well-documented solutions.	- Constant innovation and rapid development of new features and best practices.
	- Strong support from academic and industry users for traditional ML tasks.	- Widely adopted in both industry and academia for state-of-the-art deep learning.

Anngladys / AI-Tools-Assignment

Code Issues Pull requests Actions Projects Wiki Security

AI-Tools-Assignment / part2_practical / task1_classical_ml.ipynb

Anngladys Winding up 2e937b0 · 2 minutes ago

248 lines (248 loc) · 8.87 KB

Preview Code Blame Raw

In [ ]:

In [1]:

```
# Cell 1: Import Libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore') # Ignore warnings, especially for precis
```

In [2]:

```
# Cell 2: Load and Preprocess Data
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

print(f"Original Training data shape: {X_train.shape}, Labels shape: {y_tr
print(f"Original Testing data shape: {X_test.shape}, Labels shape: {y_test

# Normalize pixel values to [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape images to (height, width, channels) - MNIST is grayscale, so cha
X_train = np.expand_dims(X_train, -1) # Adds a channel dimension
X_test = np.expand_dims(X_test, -1)

print(f"\nNormalized and Reshaped Training data shape: {X_train.shape}")
print(f"Normalized and Reshaped Testing data shape: {X_test.shape}")

# One-hot encode the labels
num_classes = 10
y_train_one_hot = to_categorical(y_train, num_classes)
y_test_one_hot = to_categorical(y_test, num_classes)

print(f"One-hot encoded Training labels shape: {y_train_one_hot.shape}")
print(f"One-hot encoded Testing labels shape: {y_test_one_hot.shape}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 3
      1 # Cell 2: Load and Preprocess Data
      2 # Load the MNIST dataset
----> 3 (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_
data()
      5 print(f"Original Training data shape: {X_train.shape}, Labels shape:
{y_train.shape}")
      6 print(f"Original Testing data shape: {X_test.shape}, Labels shape:
{y_test.shape}")

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

In [ ]:

```
# Cell 3: Build the CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
```

```
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5), # Helps prevent overfitting
    Dense(num_classes, activation='softmax') # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

In [ ]:

```
# Cell 4: Train the Model
# Use GPU if available (Google Colab usually provides one)
# The training process will automatically use GPU if TF is configured for
history = model.fit(X_train, y_train_one_hot,
                    epochs=10, # You can try fewer epochs (e.g., 5) to sav
                    batch_size=64,
                    validation_split=0.1) # Use 10% of training data for v

print("\nModel training complete!")
```

In [ ]:

```
# Cell 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Cell 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Cell 6: Visualize Model Predictions
# Get 5 random sample images from the test set
sample_indices = np.random.choice(len(X_test), 5, replace=False)
sample_images = X_test[sample_indices]
sample_true_labels = y_test[sample_indices]

# Make predictions
predictions = model.predict(sample_images)
predicted_labels = np.argmax(predictions, axis=1)

plt.figure(figsize=(12, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(sample_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {sample_true_labels[i]}\nPred: {predicted_labels[i]}
    plt.axis('off')
```

AI-Tools-Assignment / part2_practical / task2_deep_learning_mnist.ipynb

Anngladys Winding up 2e937b0 · 2 minutes ago

504 lines (504 loc) · 44.2 KB

Preview Code Blame Raw

In [1]:

```
# Cell 1: Import Libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

print(f"TensorFlow Version: {tf.__version__}")
print("Libraries imported successfully!")
```

```
TensorFlow Version: 2.19.0
Libraries imported successfully!
```

In [2]:

```
# Cell 2: Load and Preprocess Data
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

print(f"Original Training data shape: {X_train.shape}, Labels shape: {y_tr
print(f"Original Testing data shape: {X_test.shape}, Labels shape: {y_test

# Normalize pixel values to [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape images to (height, width, channels) - MNIST is grayscale, so cha
X_train = np.expand_dims(X_train, -1) # Adds a channel dimension
X_test = np.expand_dims(X_test, -1)

print(f"\nNormalized and Reshaped Training data shape: {X_train.shape}")
print(f"Normalized and Reshaped Testing data shape: {X_test.shape}")

# One-hot encode the labels
num_classes = 10
y_train_one_hot = to_categorical(y_train, num_classes)
y_test_one_hot = to_categorical(y_test, num_classes)

print(f"One-hot encoded Training labels shape: {y_train_one_hot.shape}")
print(f"One-hot encoded Testing labels shape: {y_test_one_hot.shape}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
Original Training data shape: (60000, 28, 28), Labels shape: (60000,)
Original Testing data shape: (10000, 28, 28), Labels shape: (10000,)

Normalized and Reshaped Training data shape: (60000, 28, 28, 1)
Normalized and Reshaped Testing data shape: (10000, 28, 28, 1)
One-hot encoded Training labels shape: (60000, 10)
One-hot encoded Testing labels shape: (10000, 10)
```

In [3]:

```
# Cell 3: Build the CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5), # Helps prevent overfitting
    Dense(num_classes, activation='softmax') # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
c:\Users\user\OneDrive\Documents\PLP\AI_Tools_Assignment\venv\Lib\site-packa
ges\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pas
s an `input_shape`/`input_dim` argument to a layer. When using Sequential mo
dels, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Pa
conv2d (Conv2D)	(None, 26, 26, 32)	
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	
conv2d_1 (Conv2D)	(None, 11, 11, 64)	1
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	
flatten (Flatten)	(None, 1600)	
dense (Dense)	(None, 128)	20
dropout (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 10)	

Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

Non-trainable params: 0 (0.00 B)

In [4]:

```
# Cell 7: Save the Trained Model
import os

# Define the directory to save the model
model_dir = 'bonus_deployment/saved_models'
os.makedirs(model_dir, exist_ok=True) # Create the directory if it doesn't

# Define the model path
model_path = os.path.join(model_dir, 'mnist_cnn_model.h5')

# Save the entire model (architecture, weights, optimizer state)
model.save(model_path)

print(f"Model saved successfully to: {model_path}")
```

```
print("You can now find 'mnist_cnn_model.h5' inside the 'bonus_deployment/
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model saved successfully to: bonus_deployment/saved_models\mnist_cnn_model.h
5
You can now find 'mnist_cnn_model.h5' inside the 'bonus_deployment/saved_mod
els' folder.
```

In [5]:

```
# Cell 3: Build the CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5), # Helps prevent overfitting
    Dense(num_classes, activation='softmax') # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Pa
conv2d_2 (Conv2D)	(None, 26, 26, 32)	
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	
conv2d_3 (Conv2D)	(None, 11, 11, 64)	1
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	
flatten_1 (Flatten)	(None, 1600)	
dense_2 (Dense)	(None, 128)	20
dropout_1 (Dropout)	(None, 128)	
dense_3 (Dense)	(None, 10)	

Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

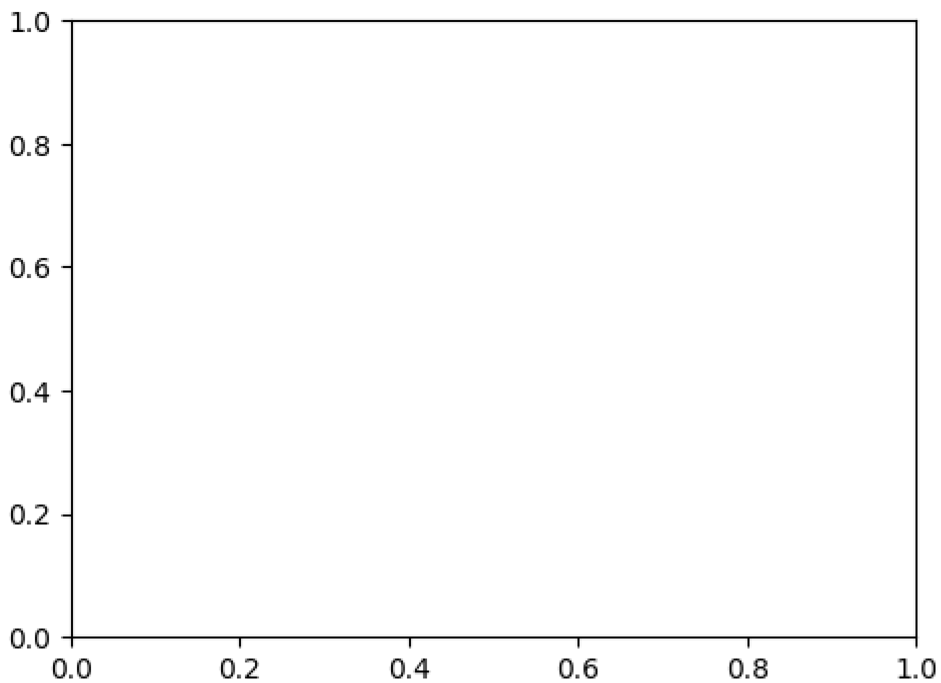
Non-trainable params: 0 (0.00 B)

In [6]:

```
# Cell 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)
```

```python
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

```
Test Loss: 2.3047
Test Accuracy: 0.0763
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 10
      8 plt.figure(figsize=(12, 4))
      9 plt.subplot(1, 2, 1)
---> 10 plt.plot(history.history['accuracy'])
     11 plt.plot(history.history['val_accuracy'])
     12 plt.title('Model Accuracy')

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



Anngladys / AI-Tools-Assignment

Code Issues Pull requests Actions Projects Wiki Security

AI-Tools-Assignment / part2_practical / task3_nlp_spacy.ipynb

Anngladys Winding up 2e937b0 · 2 minutes ago

496 lines (496 loc) · 19.4 KB

Preview Code Blame Raw

In [1]:

```python
# Cell 1: Import Libraries
import spacy

# Load the English spaCy model (ensure you've run 'python -m spacy downloa
try:
    nlp = spacy.load("en_core_web_sm")
    print("spaCy model loaded successfully!")
except OSError:
    print("SpaCy model not found. Please run 'python -m spacy download en_
    exit() # Exit if model not loaded
```

```
spaCy model loaded successfully!
```

In [2]:

```python
# Cell 2: Define Sample Review Texts
review_texts = [
    "The new iPhone 15 Pro is an amazing device. Apple has outdone themsel
    "This Samsung Galaxy S24 has a terrible battery life. Very disappointe
    "Excellent Bose QuietComfort headphones! Sound quality is superb.",
    "I bought a cheap knockoff charger, it stopped working in a week. Don'
    "The Sony PlayStation 5 is fantastic for gaming, but it's often out of
    "My new Kindle Oasis arrived quickly. It's great for reading, a truly
    "Terrible experience with this Dell XPS laptop, constant crashes."
]

print("Sample review texts defined.")
```

```
Sample review texts defined.
```

In [3]:

```python
# Cell 3: Perform Named Entity Recognition (NER)
print("--- Named Entity Recognition (NER) ---")
extracted_entities = []

for i, text in enumerate(review_texts):
    doc = nlp(text)
    entities_in_review = []
    print(f"\nReview {i+1}: \"{text}\"")
    for ent in doc.ents:
        # We're primarily interested in products, organizations, and poten
        if ent.label_ in ["ORG", "PRODUCT", "GPE", "PERSON", "NORP"]: # Ad
            entities_in_review.append({"text": ent.text, "label": ent.labe
            print(f"  - Entity: '{ent.text}' (Type: {ent.label_})")
    extracted_entities.append(entities_in_review)
```

```
--- Named Entity Recognition (NER) ---

Review 1: "The new iPhone 15 Pro is an amazing device. Apple has outdone the
mselves."
  - Entity: 'Apple' (Type: ORG)

Review 2: "This Samsung Galaxy S24 has a terrible battery life. Very disappo
inted with the brand."

Review 3: "Excellent Bose QuietComfort headphones! Sound quality is superb."
  - Entity: 'Bose QuietComfort' (Type: PERSON)

Review 4: "I bought a cheap knockoff charger, it stopped working in a week.
Don't waste your money."
```

```
Don't waste your money."

Review 5: "The Sony PlayStation 5 is fantastic for gaming, but it's often ou
t of stock."
  - Entity: 'Sony' (Type: ORG)
  - Entity: 'PlayStation 5' (Type: PRODUCT)

Review 6: "My new Kindle Oasis arrived quickly. It's great for reading, a tr
uly portable library."
  - Entity: 'Kindle Oasis' (Type: ORG)

Review 7: "Terrible experience with this Dell XPS laptop, constant crashes."
  - Entity: 'Dell XPS' (Type: ORG)
```

In [4]:

```python
# Cell 4: Analyze Sentiment (Rule-Based Approach)
print("\n--- Sentiment Analysis (Rule-Based) ---")

positive_words = ["amazing", "excellent", "superb", "fantastic", "great",
negative_words = ["terrible", "disappointed", "stopped working", "waste",

def analyze_sentiment_rule_based(text):
    text_lower = text.lower()
    pos_score = sum(1 for word in positive_words if word in text_lower)
    neg_score = sum(1 for word in negative_words if word in text_lower)

    if pos_score > neg_score:
        return "Positive"
    elif neg_score > pos_score:
        return "Negative"
    else:
        return "Neutral" # Or if pos_score == neg_score

for i, text in enumerate(review_texts):
    sentiment = analyze_sentiment_rule_based(text)
    print(f"\nReview {i+1}: \"{text}\"")
    print(f"  - Sentiment: {sentiment}")
```

```
--- Sentiment Analysis (Rule-Based) ---

Review 1: "The new iPhone 15 Pro is an amazing device. Apple has outdone the
mselves."
  - Sentiment: Positive

Review 2: "This Samsung Galaxy S24 has a terrible battery life. Very disappo
inted with the brand."
  - Sentiment: Negative

Review 3: "Excellent Bose QuietComfort headphones! Sound quality is superb."
  - Sentiment: Positive

Review 4: "I bought a cheap knockoff charger, it stopped working in a week.
Don't waste your money."
  - Sentiment: Negative

Review 5: "The Sony PlayStation 5 is fantastic for gaming, but it's often ou
t of stock."
  - Sentiment: Positive

Review 6: "My new Kindle Oasis arrived quickly. It's great for reading, a tr
uly portable library."
  - Sentiment: Positive

Review 7: "Terrible experience with this Dell XPS laptop, constant crashes."
```

```
- Sentiment: Negative
```

In [5]:

```python
# Cell 1: Import Libraries
import spacy
import pandas as pd
import random

print(f"spaCy Version: {spacy.__version__}")
print("Libraries imported successfully!")
```

```
spaCy Version: 3.8.7
Libraries imported successfully!
```

In [6]:

```python
# Cell 2: Load spaCy English Model
try:
    # Load the small English model
    nlp = spacy.load("en_core_web_sm")
    print("spaCy 'en_core_web_sm' model loaded successfully.")
except OSError:
    print("spaCy model 'en_core_web_sm' not found. Downloading...")
    spacy.cli.download("en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")
    print("spaCy model 'en_core_web_sm' downloaded and loaded successfully
```

```
spaCy 'en_core_web_sm' model loaded successfully.
```

In [7]:

```python
# Cell 3: Sample Text Data (Amazon Reviews style)
amazon_reviews = [
    "The product is excellent! Very happy with the purchase.",
    "Battery life is terrible, died after 2 hours. Very disappointed.",
    "Works as expected, good value for money. Highly recommended.",
    "This is the worst item I've ever bought. A complete waste of money.",
    "It's okay, not great, not bad. Just mediocre.",
    "Fantastic performance, totally exceeded my expectations!",
    "Wish it had more features, but it's decent for the price.",
    "The delivery was fast, but the item was damaged.",
    "Absolutely love this! The design is sleek and it's so easy to use.",
    "Received a broken one. Customer service was unhelpful."
]

print("Sample Amazon reviews loaded.")
```

```
Sample Amazon reviews loaded.
```

In [8]:

```python
# Cell 4: Tokenization, POS Tagging, and Lemmatization
print("--- Tokenization, POS Tagging, and Lemmatization ---")
for i, text in enumerate(amazon_reviews[:3]): # Process first 3 reviews fo
    doc = nlp(text)
    print(f"\nReview {i+1}: '{text}'")
    print(f"{'Token':<15} {'Lemma':<15} {'POS':<10} {'Is Alpha?':<10} {'St
    print("-" * 70)
    for token in doc:
        print(f"{str(token):<15} {token.lemma_:<15} {token.pos_:<10} {str(
```

```
--- Tokenization, POS Tagging, and Lemmatization ---

Review 1: 'The product is excellent! Very happy with the purchase.'
```

```
Review 1: 'The product is excellent! Very happy with the purchase.'
Token           Lemma           POS         Is Alpha?  Stopword?
----------------------------------------------------------------------
The             the             DET         True       True
product         product         NOUN        True       False
is              be              AUX         True       True
excellent       excellent       ADJ         True       False
!               !               PUNCT       False      False
Very            very            ADV         True       True
happy           happy           ADJ         True       False
with            with            ADP         True       True
the             the             DET         True       True
purchase        purchase        NOUN        True       False
.               .               PUNCT       False      False

Review 2: 'Battery life is terrible, died after 2 hours. Very disappointed.'
Token           Lemma           POS         Is Alpha?  Stopword?
----------------------------------------------------------------------
Battery         battery         NOUN        True       False
life            life            NOUN        True       False
is              be              AUX         True       True
terrible        terrible        ADJ         True       False
,               ,               PUNCT       False      False
died            die             VERB        True       False
after           after           ADP         True       True
2               2               NUM         False      False
hours           hour            NOUN        True       False
.               .               PUNCT       False      False
Very            very            ADV         True       True
disappointed    disappointed    ADJ         True       False
.               .               PUNCT       False      False

Review 3: 'Works as expected, good value for money. Highly recommended.'
Token           Lemma           POS         Is Alpha?  Stopword?
----------------------------------------------------------------------
Works           work            NOUN        True       False
as              as              SCONJ       True       True
expected        expect          VERB        True       False
,               ,               PUNCT       False      False
good            good            ADJ         True       False
value           value           NOUN        True       False
for             for             ADP         True       True
money           money           NOUN        True       False
.               .               PUNCT       False      False
Highly          highly          ADV         True       False
recommended     recommend       VERB        True       False
.               .               PUNCT       False      False
```

In [9]:

```python
# Cell 5: Named Entity Recognition (NER)
print("\n--- Named Entity Recognition (NER) ---")
for i, text in enumerate(amazon_reviews):
    doc = nlp(text)
    if doc.ents:
        print(f"\nReview {i+1}: '{text}'")
        for ent in doc.ents:
            print(f"  Entity: {ent.text}, Type: {ent.label_}, SpaCy Explan
    else:
        print(f"\nReview {i+1}: '{text}' - No entities found.")
```

```
--- Named Entity Recognition (NER) ---

Review 1: 'The product is excellent! Very happy with the purchase.' - No ent
ities found.
```

```
Review 2: 'Battery life is terrible, died after 2 hours. Very disappointed.'
  Entity: 2 hours, Type: TIME, SpaCy Explanation: Times smaller than a day

Review 3: 'Works as expected, good value for money. Highly recommended.' - N
o entities found.

Review 4: 'This is the worst item I've ever bought. A complete waste of mone
y.' - No entities found.

Review 5: 'It's okay, not great, not bad. Just mediocre.' - No entities foun
d.

Review 6: 'Fantastic performance, totally exceeded my expectations!'
  Entity: Fantastic, Type: NORP, SpaCy Explanation: Nationalities or religio
us or political groups

Review 7: 'Wish it had more features, but it's decent for the price.' - No e
ntities found.

Review 8: 'The delivery was fast, but the item was damaged.' - No entities f
ound.

Review 9: 'Absolutely love this! The design is sleek and it's so easy to us
e.' - No entities found.

Review 10: 'Received a broken one. Customer service was unhelpful.' - No ent
ities found.
```

In [10]:

```python
# Cell 6: Basic Rule-Based Sentiment Analysis (Illustrative - very simple)
print("\n--- Basic Rule-Based Sentiment Analysis (Illustrative) ---")

positive_words = ["excellent", "happy", "good", "recommended", "fantastic"
negative_words = ["terrible", "disappointed", "worst", "waste", "mediocre"

def simple_sentiment(text):
    doc = nlp(text.lower()) # Process lowercase text
    sentiment_score = 0
    for token in doc:
        if token.text in positive_words:
            sentiment_score += 1
        elif token.text in negative_words:
            sentiment_score -= 1
    if sentiment_score > 0:
        return "Positive"
    elif sentiment_score < 0:
        return "Negative"
    else:
        return "Neutral"

for i, review in enumerate(amazon_reviews):
    sentiment = simple_sentiment(review)
    print(f"Review {i+1}: '{review}'\n  Sentiment: {sentiment}\n")

print("\nNote: This is a very simplistic rule-based sentiment analysis.")
print("It lacks context understanding, sarcasm detection, and nuances. For
print("Review: 'This is great, another broken item!' (Should be Negative)"
doc_sarcasm = nlp("This is great, another broken item!")
print(f"  Simple rule-based analysis: {simple_sentiment(str(doc_sarcasm))}
print("\nAdvanced NLP (like machine learning models or deep learning) is n
```

```
--- Basic Rule-Based Sentiment Analysis (Illustrative) ---
Review 1: 'The product is excellent! Very happy with the purchase.'
```

```
Review 1: 'The product is excellent! Very happy with the purchase.'
  Sentiment: Positive

Review 2: 'Battery life is terrible, died after 2 hours. Very disappointed.'
  Sentiment: Negative

Review 3: 'Works as expected, good value for money. Highly recommended.'
  Sentiment: Positive

Review 4: 'This is the worst item I've ever bought. A complete waste of mone
y.'
  Sentiment: Negative
```

Anngladys / AI-Tools-Assignment

Code Issues Pull requests Actions Projects Wiki Security

AI-Tools-Assignment / part3_ethics_optimization / **debug_tensorflow.py**

Anngladys feat: Completed theoretical answers and debugged TensorFlow script

18fadde · 1 hour ago

55 lines (45 loc) · 2.51 KB

Code Blame Raw

```
# debug_tensorflow.py (CORRECTED VERSION)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Input # Import Input layer
from tensorflow.keras.optimizers import Adam
import numpy as np

print("Attempting to run corrected code...")

# --- FIXED BUGS ---

# Bug 1 & 5 FIXED: Correct data generation/types and shape for MNIST-like structure
# Assuming we want 28x28 grayscale images, 10 classes, and float32 type
num_samples = 100
image_height = 28
image_width = 28
num_channels = 1 # Grayscale images have 1 channel
num_classes = 10 # Model output will be 10 classes (digits 0-9)

# Generate dummy image data (pixel values between 0 and 1)
X_train_corrected = np.random.rand(num_samples, image_height, image_width, num_channels).a
# Generate dummy integer labels (0 to 9)
y_train_corrected = np.random.randint(0, num_classes, num_samples)

print(f"Generated X_train_corrected shape: {X_train_corrected.shape}")
print(f"Generated y_train_corrected shape: {y_train_corrected.shape}")

# Bug 2 FIXED: Model architecture with explicit Input layer and correct output Dense layer
model = Sequential([
    Input(shape=(image_height, image_width, num_channels)), # Explicitly define input shap
    Flatten(), # Flatten the 28x28x1 image into a 784-element vector
    Dense(128, activation='relu'), # Hidden layer with ReLU activation
    Dense(num_classes, activation='softmax') # Output layer for 10 classes with softmax fo
])

```

```
# Bug 3 & 4 FIXED: Correct loss function for integer labels (sparse_categorical_crossentro
# Adam optimizer and accuracy metric are appropriate
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', # Corrected: use sparse_categorical_
              metrics=['accuracy'])

print("\nModel compiled successfully.")
model.summary() # Print model summary to verify architecture

# Attempt training with corrected data
try:
    print("\nStarting dummy training for 1 epoch...")
    # Use verbose=0 to suppress per-batch output, just show epoch summary
    model.fit(X_train_corrected, y_train_corrected, epochs=1, batch_size=32, verbose=1)
    print("\nCorrected model training attempted and succeeded for 1 epoch!")
except Exception as e:
    print(f"\nError during training (this should not happen with corrected code): {e}")
    print("Please double-check the pasted code and ensure your venv is active.")

print("\n--- End of Corrected Code Execution ---")
```

Ethical Considerations in AI Models

Potential Biases

MNIST Handwritten Digits Model

While less prone to social biases like those found in language models, the MNIST dataset and models trained on it can exhibit **data collection bias**.

- **Bias Source:** If the dataset predominantly features handwriting from a specific demographic (e.g., adults, people from a certain region, right-handed individuals), the model might perform suboptimally on handwriting samples from underrepresented groups (e.g., children, left-handed individuals, different cultural writing styles). This could lead to unequal performance and potentially exclude users whose handwriting doesn't fit the 'norm' the model was trained on.

Amazon Product Reviews Model (Sentiment and NER)

This model, particularly the sentiment analysis, is highly susceptible to various biases.

- **Algorithmic Bias in NER:** The Named Entity Recognition (NER) model, even if using a pre-trained spaCy model, is trained on general text. It might struggle to identify niche product names, brand variations, or specific jargon common in product reviews from particular industries or communities. This could lead to **under-recognition** of entities relevant to certain products or user groups.
- **Sentiment Analysis Bias (Rule-Based):**
  - **Vocabulary Bias:** Our simple rule-based system relies on predefined positive/negative word lists. It may fail to capture sentiment expressed

through sarcasm, irony, slang, regional dialects, or nuanced expressions. For instance, "sick" can be negative in health contexts but positive as slang ("that's sick!").
    - **Product Category Bias:** A word might have different sentiment implications depending on the product. "Hot" is positive for a new gadget but negative for a laptop that overheats. The rule-based approach won't differentiate this contextually.
    - **Demographic Bias:** If certain user demographics (e.g., younger users, specific cultural groups, non-native English speakers) use language patterns, abbreviations, or emotional expressions differently, the model could misclassify their sentiment, leading to an inaccurate representation of their opinions.

Mitigation Strategies

Mitigating Bias with Tools (or approaches)

- **TensorFlow Fairness Indicators (for MNIST - Conceptual Application):**

    - While direct application for raw image data like MNIST is complex, if metadata about the writers (e.g., age group, gender, region) were available and associated with MNIST samples, TensorFlow Fairness Indicators could be used.
    - **How:** One would define sensitive groups based on this metadata. Fairness Indicators would then measure and visualize performance metrics (e.g., accuracy, false positive rates) across these groups. If disparities are found, it would signal a need for more diverse data collection or re-weighting of samples during training to ensure equitable performance. *For this specific assignment, its direct application for MNIST is limited without additional metadata.*

- **spaCy's Rule-Based Systems and Extensions (for Amazon Reviews):**

    - **Mitigating NER Bias:**
        - **Custom Rules/Matchers:** If the statistical NER model consistently misses specific product names or brand patterns (e.g., "XYZ-Pro Max"), we can augment it with spaCy's rule-based `Matcher` to explicitly recognize these. This allows for precise extraction of known entities regardless of the statistical model's performance on them.
        - **Fine-tuning:** For a more robust solution, fine-tuning a spaCy NER model on a diverse, domain-specific dataset of product reviews (annotated for entities) would significantly improve its performance and reduce bias towards general text patterns.
    - **Mitigating Sentiment Bias:**

- **Expanded & Contextual Lexicons:** Enhance the `positive_words` and `negative_words` lists to be more comprehensive and domain-specific. Incorporate multi-word expressions (e.g., "great value", "not working well").
- **Negation Handling:** Implement simple rules to reverse sentiment for negated words (e.g., "not good" should be negative). SpaCy's dependency parser could help identify negation tokens.
- **Part-of-Speech (POS) and Dependency Parsing:** Leverage spaCy's capabilities to understand the grammatical context. For instance, "hot" modifying a positive noun like "deal" vs. "hot" modifying a negative noun like "surface temperature." This moves beyond simple keyword matching to more sophisticated contextual analysis.
- **User Feedback & Iteration:** Continuously collect user feedback on sentiment classifications and use it to refine the rule-based system or train more advanced models.
- **Ensemble Approaches:** Combine the rule-based system with a pre-trained general sentiment model (like one from Hugging Face Transformers) and analyze where they differ, learning from their disagreements.