Ethics and Optimization.

Machine learning models trained on datasets like MNIST and Amazon Product Reviews can inherit biases from the underlying data. For MNIST, the data primarily represents handwriting from a limited demographic, potentially leading to poor performance on digits written by people from different regions or age groups. In Amazon Reviews, biases can arise from class imbalances (e.g., more positive than negative reviews), overrepresentation of popular brands (like Apple or Sony), and cultural or linguistic variations in how people express sentiment. These biases can affect the accuracy of sentiment analysis and named entity recognition (NER), leading to unfair or skewed outcomes.

Tools like TensorFlow Fairness Indicators and spaCy's rule-based systems help identify and reduce such biases. TensorFlow Fairness Indicators allow developers to visualize and compare model performance across different subgroups, making it easier to detect and address disparities. On the other hand, spaCy's rule-based NER can supplement the statistical model by manually adding patterns for underrepresented brands or products, improving coverage and fairness. Together, these tools support more equitable and accurate model development.

Troubleshooting.

When debugging a TensorFlow script with errors such as dimension mismatches or incorrect loss functions, the first step is to closely inspect the shape of the input data and labels. Use print statements or debugging tools to verify that the training and testing data align with the model's expected input and output shapes. For example, convolutional neural networks expect 4D input tensors (batch, height, width, channels), so grayscale images like those in MNIST should be reshaped accordingly. It's also important to ensure that labels are in the right format: use integer labels for sparse_categorical_crossentropy and one-hot encoded vectors for categorical_crossentropy. Running model.summary() can help visualize layer connections and pinpoint shape mismatches between layers.

Next, confirm that the model's final layer and the loss function are correctly matched. A common mistake is using a softmax output with the wrong label encoding or an incompatible loss function. Also, examine whether metrics like accuracy are suitable for your task—avoid using them for regression problems. Beyond architecture, issues can arise during training if the batch size is too large or if preprocessing steps are inconsistent between training and validation data. TensorFlow's built-in debugging functions such as tf.debugging.assert_shapes() and verbose mode during training can help isolate issues. Finally, if errors persist, simplify the script into a minimal working example with synthetic data to test smaller components of the model in isolation. This step-by-step approach helps identify and resolve specific bugs while reinforcing your understanding of model mechanics.