TensorFlow

Recurrent Neural Networks using Tensor Flow

Recurrent Neural Networks

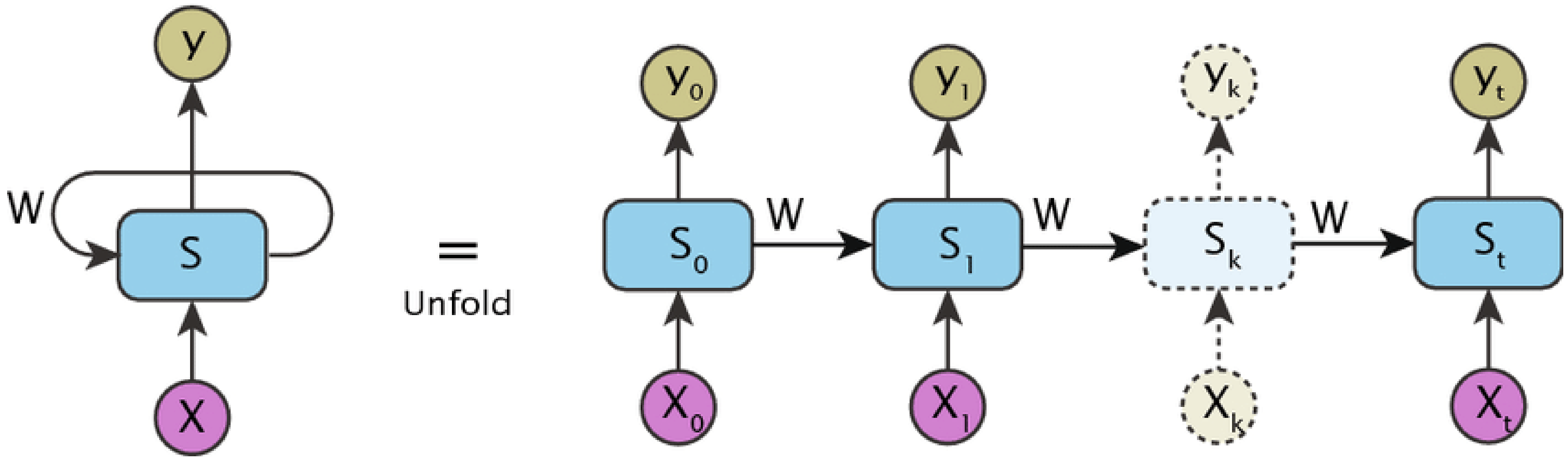
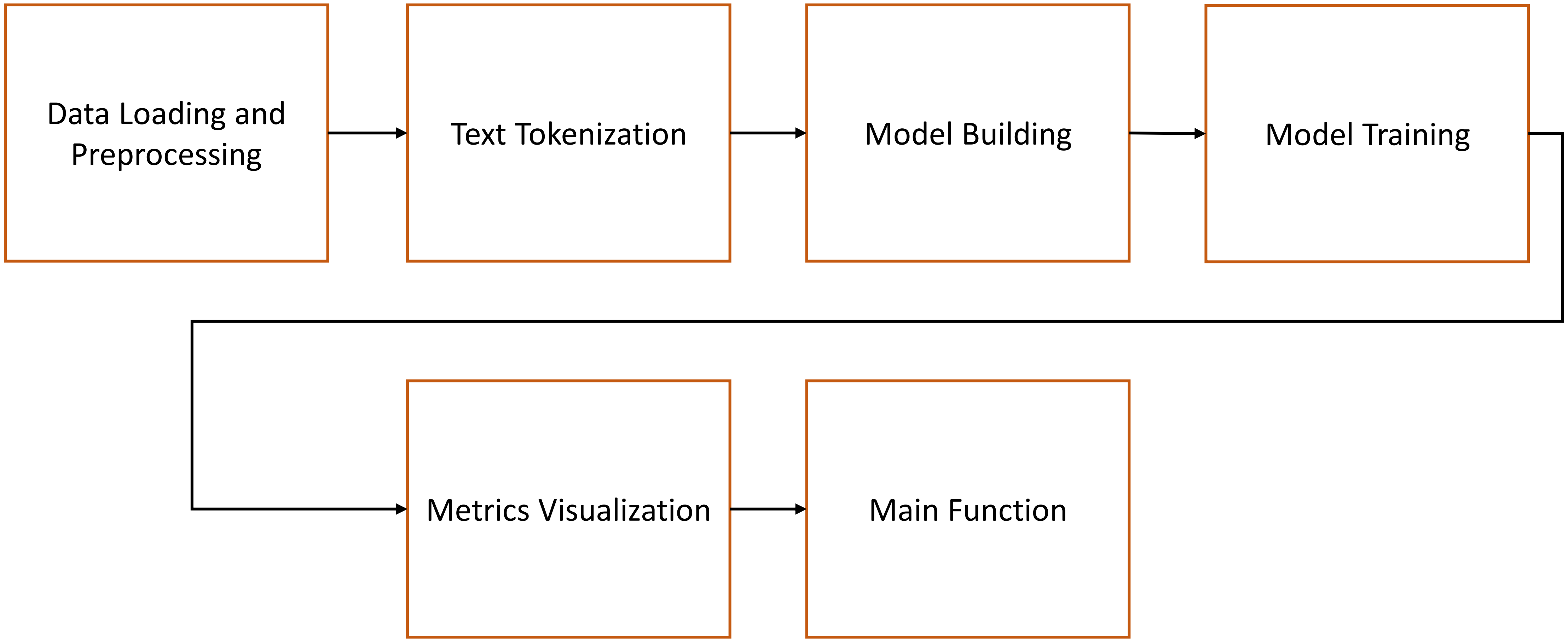


Image Processing Pipeline



Data Loading & Preprocessing

- For reading a CSV file containing text and sentiment labels, and encodes the labels.

```
# Load and preprocess data
def load_data(file_path):
    df = pd.read_csv(file_path)
    texts = df['text'].values

    # Encode sentiment labels
    label_encoder = LabelEncoder()
    labels = label_encoder.fit_transform(df['sentiment'])

    return texts, labels, label_encoder
```

Data Preprocessing

- For tokenizing and pads the input sequences

```
# Tokenize and pad sequences
def preprocess_text(texts, max_words=1000, max_len=100):
    tokenizer = Tokenizer(num_words=max_words)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    padded_sequences = pad_sequences(sequences, maxlen=max_len)

    return padded_sequences, tokenizer
```

Model Building

- For creating an RNN model with an Embedding layer, a SimpleRNN layer, and a Dense output layer

```
# Build the RNN model
def build_model(max_words, max_len, embedding_dim=50, rnn_units=64):

    model = Sequential([
        Embedding(max_words, embedding_dim, input_length=max_len),
        SimpleRNN(rnn_units),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

Model Training

- For training the model and returns the training history.

```
# Train the model
def train_model(model, X_train, y_train, X_val, y_val, epochs=10, batch_size=32):
    history = model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=epochs,
                        batch_size=batch_size,
                        verbose=1)

    return history
```

Performance Evaluation

```
# Plot training history
def plot_history(history):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

    # Accuracy plot
    ax1.plot(history.history['accuracy'], label='Training Accuracy')
    ax1.plot(history.history['val_accuracy'], label='Validation Accuracy')
    ax1.set_title('Model Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()

    # Loss plot
    ax2.plot(history.history['loss'], label='Training Loss')
    ax2.plot(history.history['val_loss'], label='Validation Loss')
    ax2.set_title('Model Loss')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()

    plt.tight_layout()
    plt.show()
```

Main Function

```
# Main function
def main():
    # Load and preprocess data
    texts, labels, label_encoder = load_data('sentiment_data.csv')
    X, tokenizer = preprocess_text(texts)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

    # Build and train the model
    model = build_model(max_words=1000, max_len=100)
    history = train_model(model, X_train, y_train, X_val, y_val)

    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Test accuracy: {test_accuracy:.4f}")

    # Plot training history
    plot_history(history)
```

Exercise: Time Series Prediction with RNN

- Modify the provided RNN sentiment analysis code to create a time series prediction model for the Air Quality dataset. Your task is to predict the concentration of CO for the next hour based on the previous 24 hours of sensor data.
- Dataset: UCI Air Quality Dataset
- URL: https://archive.ics.uci.edu/ml/datasets/Air+Quality.

To-do list

- Data Loading and Preprocessing:
  - Load the dataset from the URL using pandas
  - Select relevant features (you may start with 'CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)')
  - Handle missing values appropriately
  - Normalize the data

Model Training

- Sequence Preparation:
  - Create sequences of 24 time steps (hours) as input
  - Use the next hour's 'CO(GT)' value as the target
- Model Architecture:
  - Modify the RNN architecture to handle multivariate input
  - Experiment with different RNN types (SimpleRNN, LSTM, GRU)
  - Adjust the number of layers and units as needed 4.
- Training and Evaluation:
  - Split the data into training, validation, and test sets
  - Train the model and evaluate its performance
  - Use appropriate metrics for regression (e.g., Mean Squared Error, Mean Absolute Error)
- Visualization:
  - Plot the actual vs predicted values for a subset of the test data
  - Visualize the training and validation loss over epochs