End-to-End Inventory Management Model with PyTorch

February 17, 2025

1 Overview

This document provides a detailed explanation of the `E2EModel`, a neural network model implemented using PyTorch for end-to-end inventory management. The model is designed to predict optimal reorder quantities, forecast demand, and manage inventory levels efficiently. Below, we break down the architecture, layers, and functionality of the model.

2 Model Architecture

2.1 Embedding Layers

The model starts by processing categorical features (i.e. SKU and Store IDs) using embedding layers. These layers convert categorical IDs into dense vectors, which are then used as inputs to the network.

```
self.embeddings = nn.ModuleList([
    nn.Embedding(m + 1, math.ceil(math.log(m + 1)))
    for m in max_cat_id
])
```

2.2 Dilated Convolution Layers

The model uses dilated convolutions to process sequential input data (i.e. inventory levels, demand, lead time). Dilated convolutions allow the model to capture long-range dependencies in the data.

```
self.conv_layers = nn.ModuleList()
self.conv_layers.append(
    nn.Conv1d(
        n_dyn_fea,
        n_filters,
        kernel_size,
        padding='same',
        dilation=1
    )
)
for i in range(1, n_dilated_layers):
    self.conv_layers.append(
        nn.Conv1d(
            n_filters,
            n_filters,
            kernel_size,
            padding='same',
            dilation=2 ** i
        )
    )
```

2.3 Final Convolution Layer

After the dilated convolutions, a final convolution layer is applied to reduce the dimensionality of the feature maps.

```
self.final_conv = nn.Conv1d(n_filters * 2 if n_dilated_layers > 1 else n_filters, 8, 1)
```

2.4 Dense Layers

The output from the convolutional layers is flattened and concatenated with the embedded categorical features. This combined data is then passed through dense layers to produce the final predictions.

```
self.dense1 = nn.Linear(conv_out_size + embed_total_size, 64)
self.output_layer = nn.Linear(64, n_outputs)
```

2.5 VLT (Lead Time) Processing

The model processes lead time (`vlt_input`) separately using dense layers. This allows the model to capture the impact of lead time on inventory decisions.

```
self.vlt_dense1 = nn.Linear(seq_len + embed_total_size, 16)
self.vlt_dense2 = nn.Linear(16, 1)
```

2.6 Reorder Point and Initial Stock Layers

The model includes additional layers to process reorder points and initial stock levels. These layers help the model make more accurate predictions about when to reorder and how much to reorder.

```
self.layer3_dense = nn.Linear(64 + 16 + seq_len, 32)
self.layer4_dense1 = nn.Linear(32 + seq_len, 64)
self.layer4_dense2 = nn.Linear(64, 1)
```

3 Custom Loss Function

The model uses a custom loss function to optimize the predictions. The loss function combines the mean squared error (MSE) for demand forecast, reorder quantity, and lead time predictions.

```
def custom_loss(y_true, y_pred):
    df_true, layer4_true, vlt_true = y_true
    df_pred, layer4_pred, vlt_pred = y_pred

    mse_df = F.mse_loss(df_pred, df_true)
    mse_layer4 = F.mse_loss(layer4_pred, layer4_true)
    mse_vlt = F.mse_loss(vlt_pred, vlt_true)

    loss = 0.05 * mse_df + 0.9 * mse_layer4 + 0.05 * mse_vlt
    return loss
```

4 Usage

To use the `E2EModel`, follow these steps:

4.1 Initialize the Model

```
model = E2EModel(
    seq_len=seq_len,
    n_dyn_fea=len(dynamic_features),
    n_outputs=forecast_horizon,
    n_dilated_layers=5,
    kernel_size=2,
    n_filters=3,
    dropout_rate=DROPOUT_RATE,
    max_cat_id=[MAX_SKU_ID, MAX_STORE_ID]
).to(device)
```

4.2 Define the Optimizer

```
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

4.3 Train the Model

```
train_losses, df_pred = train_model(model, train_loader, val_loader, optimizer, device,
                                    EPOCHS, custom_loss)
```

4.4 Evaluate the Model

```
test_loss = evaluate_model(model, val_loader, device, custom_loss)
```

4.5 Visualize Results

```
plot_inventory_level(train_time, train_inventory, test_time, test_inventory)
plot_demand_forecast(train_time, train_demand, test_time, test_demand,
                                    forecasted_demand_reshaped)
```