(a) A Vanilla Recurrent Neural Network (RNN) has the following settings. 香草递归神经网络（RNN）具有以下设置。

Initial hidden state, $\mathbf{h}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, 初始隐藏状态，

Hidden state weight matrix, $\mathbf{W}_{hh} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$, 隐状态权矩阵，

Input weight matrix, $\mathbf{W}_{xh} = \begin{bmatrix} 0.5 & 0.2 \\ 0.2 & 0.1 \end{bmatrix}$, 输入权矩阵，

Output weight matrix, $\mathbf{W}_{hy} = \begin{bmatrix} 0.1 & 0.4 \end{bmatrix}$. 输出权矩阵

Assume no bias is used in the computation of the RNN. 假设在RNN的计算中没有使用偏差。

A 2-timestep input is given by $\mathbf{x} = [\mathbf{x}_1 \quad \mathbf{x}_2]$ where $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$. 一个2时间步长的输入由

(i) Find the hidden state $\mathbf{h}_1$ at timestep $t = 1$. 找出时间步长t=1时的隐藏状态h。

(ii) Find the output y_1 at timestep $t = 1$. 求时间步长t=1时的输出y。

(iii) Find the output y_2 at timestep $t = 2$. 求时间步长t=2时的输出y。

用户希望使用Vanilla RNN根据以下输入特征来预测股票的价格：

过去5天的价格、交易量和股票的出价。

建议对上述RNN结构进行更改以执行股价预测。

(iv) A user would like to use a Vanilla RNN to predict the price of a share based on the following input features: the price, the trade volume, and the bid number of the share for the past 5 days. Suggest the change(s) that should be made to the RNN structure above to perform the share price prediction.

(15 Marks)

(b) A user would like to develop an image classification application using a model that can achieve good accuracy and uses attention mechanism when performing classification. He is considering the following 3 candidate models: (i) VGG, (ii) Vision Transformer (ViT), and (iii) Long Short-Term Memory (LSTM). State which model is most likely going to meet the user's need and briefly justify your answer.

(5 Marks)

用户希望开发一个图像分类应用程序，

该应用程序使用的模型在执行分类时能够达到

良好的准确率并使用注意机制。

他正在考虑以下3种候选模型：

（i）VGG，

（ii）视觉变压器（ViT）

和（iii）长短期记忆（LSTM）。

说明哪个模型最有可能满足用户的需求，

并简要地证明你的答案。

Solution 解决方案

(a) (i)

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t$$

At timestep t=1:

$$\mathbf{h}_1 = \tanh(\mathbf{W}_{hh}\mathbf{h}_0 + \mathbf{W}_{xh}\mathbf{x}_1)$$

$$= \tanh\left(\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.2 \\ 0.2 & 0.1 \end{bmatrix}\begin{bmatrix} 3 \\ 4 \end{bmatrix}\right)$$

$$= \tanh\left(\begin{bmatrix} 2.3 \\ 1.0 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.9801 \\ 0.7616 \end{bmatrix}$$

(ii)

$$\mathbf{y}_1 = \mathbf{W}_{hy}\mathbf{h}_1 = \begin{bmatrix} 0.1 & 0.4 \end{bmatrix}\begin{bmatrix} 0.9801 \\ 0.7616 \end{bmatrix} = 0.4026$$

(iii)

At timestep t=2:

$$\mathbf{h}_2 = \tanh(\mathbf{W}_{hh}\mathbf{h}_1 + \mathbf{W}_{xh}\mathbf{x}_2)$$

$$= \tanh\left(\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}\begin{bmatrix} 0.9801 \\ 0.7616 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.2 \\ 0.2 & 0.1 \end{bmatrix}\begin{bmatrix} 1 \\ 6 \end{bmatrix}\right)$$

$$= \tanh\left(\begin{bmatrix} 1.9503 \\ 1.3987 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.9603 \\ 0.8851 \end{bmatrix}$$

$$\mathbf{y}_2 = \mathbf{W}_{hy}\mathbf{h}_2 = \begin{bmatrix} 0.1 & 0.4 \end{bmatrix}\begin{bmatrix} 0.9603 \\ 0.8851 \end{bmatrix} = 0.4501$$

思考 17 秒

To adapt the Vanilla Recurrent Neural Network (RNN) structure for predicting the share price based on the given features (price, trade volume, and bid number over the past 5 days), the following changes should be made:

为了适应普通循环神经网络 (RNN) 结构以根据给定特征（过去 5 天的价格、交易量和出价数量）预测股价，应进行以下更改：

1. **Input Layer Modification:输入层修改**:
   - **Adjust Input Dimensions**: Modify the input layer to accept a feature vector of size 3 at each time step, representing the price, trade volume, and bid number.

     调整输入维度：修改输入层以在每个时间步接受大小为 3 的特征向量，表示价格、交易量和出价数量。
   - **Sequence Length Adjustment**: Set the sequence length to 5 to account for the past 5 days' data.**序列长度调整**：将序列长度设置为 5 以考虑过去 5 天的数据。

   Implementation:执行:

   python 复制代码
   ```python
   # Example in PyTorch
   input_size = 3   # Price, Volume, Bid Number
   sequence_length = 5  # Past 5 days
   ```

2. **RNN Architecture Adjustment:RNN架构调整**:
   - **Many-to-One Configuration**: Configure the RNN to have a many-to-one architecture, where it processes the input sequence and produces an output only at the final time step.

     多对一配置：将 RNN 配置为多对一架构，处理输入序列并仅在最后时间步产生输出。
   - **Hidden Layer Size**: Determine an appropriate size for the hidden layer(s) to capture temporal dependencies.**隐藏层大小**：确定隐藏层的适当大小以捕获时间依赖性。

   Implementation:执行:

   python 复制代码
   ```python
   # Example in PyTorch
   hidden_size = 128  # Can be tuned
   num_layers = 1     # Number of RNN layers
   rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
   ```

3. **Output Layer Modification:输出层修改**:

   - **Single Output Neuron**: Modify the output layer to have a single neuron to predict the continuous share price.

     单输出神经元：修改输出层以使用单个神经元来预测连续股价。

   - **Linear Activation Function**: Use a linear activation function (or no activation function) at the output layer since price prediction is a regression task.

     线性激活函数：在输出层使用线性激活函数（或不使用激活函数），因为价格预测是一项回归任务。

   Implementation:执行:

   python 复制代码

```python
# Example in PyTorch
fc = nn.Linear(hidden_size, 1)  # Output layer with one neuron
```

4. **Loss Function and Optimization:损失函数和优化**:

   - **Appropriate Loss Function**: Use a regression loss function such as Mean Squared Error (MSE) to measure the difference between the predicted and actual prices.

     适当的损失函数：使用回归损失函数，例如均方误差（MSE）来衡量预测价格和实际价格之间的差异。

   - **Optimizer Selection**: Choose an optimizer like Adam or SGD to update the network weights during training.

     优化器选择：选择 Adam 或 SGD 等优化器在训练期间更新网络权重。

   Implementation:执行:

   python 复制代码

```python
# Example in PyTorch
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=0.001)
```

5. **Activation Functions:激活函数**:

   - **Hidden Layers**: Use activation functions like Tanh or ReLU in the hidden layers to introduce non-linearity.

     隐藏层：在隐藏层中使用 Tanh 或 ReLU 等激活函数来引入非线性。

   - **Output Layer**: No activation function is needed at the output layer for regression tasks.**输出层**：回归任务的输出层不需要激活函数。

   Implementation:执行:

   python 复制代码

   ```python
   # The RNN module in PyTorch includes activation functions internally
   ```

6. **Data Preprocessing:数据预处理**:

   - **Normalization/Standardization:** Normalize or standardize the input features to improve training stability and performance.

     归一化/标准化：对输入特征进行归一化或标准化，以提高训练稳定性和性能。

   - **Feature Scaling**: Ensure that all input features are on a similar scale.

     特征缩放：确保所有输入特征都具有相似的缩放比例。

   Implementation:执行:

   python 复制代码

   ```python
   # Example using sklearn
   from sklearn.preprocessing import StandardScaler
   scaler = StandardScaler()
   X_scaled = scaler.fit_transform(X)
   ```

7. **Training Procedure:培训程序**:

- **Backpropagation Through Time (BPTT)**: Train the RNN using BPTT to adjust weights based on the temporal dependencies in the data.

  随时间反向传播 (BPTT)： 使用 BPTT 训练 RNN，以根据数据中的时间依赖性调整权重。

- **Batch Processing**: Use appropriate batch sizes and shuffle the data to improve generalization.**批处理**：使用适当的批大小并打乱数据以提高泛化能力。

Implementation:执行:

python 复制代码

```python
# Training loop in PyTorch
for epoch in range(num_epochs):
    for batch in data_loader:
        # Forward pass
        outputs, _ = rnn(batch_inputs)
        predictions = fc(outputs[:, -1, :])  # Only use the output from the last time

        # Compute loss
        loss = criterion(predictions, batch_targets)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

8. **Adjusting the Original RNN Structure:调整原来的RNN结构**:

- **Remove Unnecessary Outputs**: If the original RNN outputs at every time step, modify it to output only at the final time step.

  删除不必要的输出：如果原始RNN在每个时间步都有输出，请将其修改为仅在最后一个时间步输出。

- **Network Depth and Complexity**: Adjust the number of layers and neurons to prevent overfitting or underfitting.

  网络深度和复杂度：调整层数和神经元数量，防止过拟合或欠拟合。

9. **Evaluation Metrics:评估指标**:

- **Performance Monitoring**: Use evaluation metrics suitable for regression, such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE), to assess model performance.

  性能监控：使用适合回归的评估指标，例如均方根误差 (RMSE) 或平均绝对误差 (MAE)，来评估模型性能。

Implementation:执行:

python 复制代码

```python
# After training
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(actual_prices, predicted_prices)
rmse = np.sqrt(mse)
```

Summary of Changes:变更摘要:

- **Input Layer**: Modify to accept a 3-dimensional feature vector over a sequence length of 5.**输入层**：修改为接受序列长度为 5 的 3 维特征向量。
- **RNN Configuration**: Adjust to a many-to-one architecture with appropriate hidden layer size.**RNN 配置**：调整为具有适当隐藏层大小的多对一架构。
- **Output Layer**: Change to a single neuron with a linear activation function for regression output.**输出层**：改为具有线性激活函数的单个神经元，用于回归输出。
- **Loss Function**: Use Mean Squared Error (MSE) suitable for continuous value prediction. **损失函数**：使用适合连续值预测的均方误差（MSE）。
- **Activation Functions**: Use suitable activation functions in hidden layers; no activation in the output layer.**激活函数**：在隐藏层中使用合适的激活函数；输出层没有激活。
- **Data Preprocessing**: Normalize or standardize input features for better model performance.**数据预处理**：规范化或标准化输入特征以获得更好的模型性能。
- **Training Procedure**: Implement BPTT and adjust the training loop to accommodate the changes.**培训程序**：实施 BPTT 并调整培训循环以适应变化。

By making these modifications, the Vanilla RNN structure will be tailored to predict the share price based on the provided features over the specified time sequence.

通过进行这些修改，Vanilla RNN 结构将被定制为根据指定时间序列上提供的特征来预测股价。

Solution 解决方案

Answer: (ii) Vision Transformer (ViT).

Vision Transformer (ViT) is based on Transformer architecture that uses attention mechanism and can achieve very good accuracy.

VGG is a CNN that does not leverage on global attention. It uses convolutional layers to progressively extract higher-level abstraction features.

LSTM is a model that uses memory to analyse sequential data. Hence it is not suitable for image classification application.

答案：（ii）视觉变压器（ViT）。

视觉变压器（Vision Transformer，ViT）

是基于Transformer架构，利用注意机制，可以达到很好的精度。

VGG是一个不利用全局关注的CNN。

它使用卷积层逐步提取更高级别的抽象特征。

LSTM是一种使用内存分析顺序数据的模型。

因此不适合用于图像分类。

The **Vision Transformer (ViT)** is the most suitable model for the user's needs.

Vision Transformer (ViT)是最适合用户需求的模型。

Justification:理由：

- **Use of Attention Mechanism:** ViT leverages the Transformer architecture, which is built upon self-attention mechanisms. This allows the model to capture long-range dependencies and relationships within image data effectively.

  使用注意力机制： ViT 利用基于自注意力机制的 Transformer 架构。这使得模型能够有效地捕获图像数据中的远程依赖性和关系。

- **Image Classification Capability:** ViT is specifically designed for image classification tasks and has demonstrated strong performance on benchmarks, making it well-suited for developing image classification applications.

  图像分类能力： ViT专为图像分类任务而设计，在基准测试中表现出强大的性能，非常适合开发图像分类应用。

- **Comparison with Other Models:与其他型号比较：**
  - **VGG:** While VGG is a powerful convolutional neural network for image classification, it does not incorporate attention mechanisms.

    VGG： 虽然 VGG 是一个强大的用于图像分类的卷积神经网络，但它没有包含注意力机制。

  - **LSTM:** LSTMs are tailored for sequential data and are not typically used for image classification. Although they can utilize attention mechanisms, they are less effective for processing image data compared to ViT.

    LSTM： LSTM 是为序列数据量身定制的，通常不用于图像分类。尽管它们可以利用注意力机制，但与 ViT 相比，它们处理图像数据的效率较低。