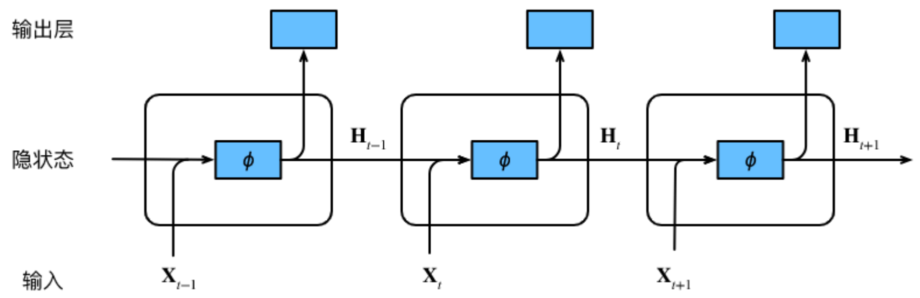
MY LSTM WORKFLOW

Present by 邓廷钦

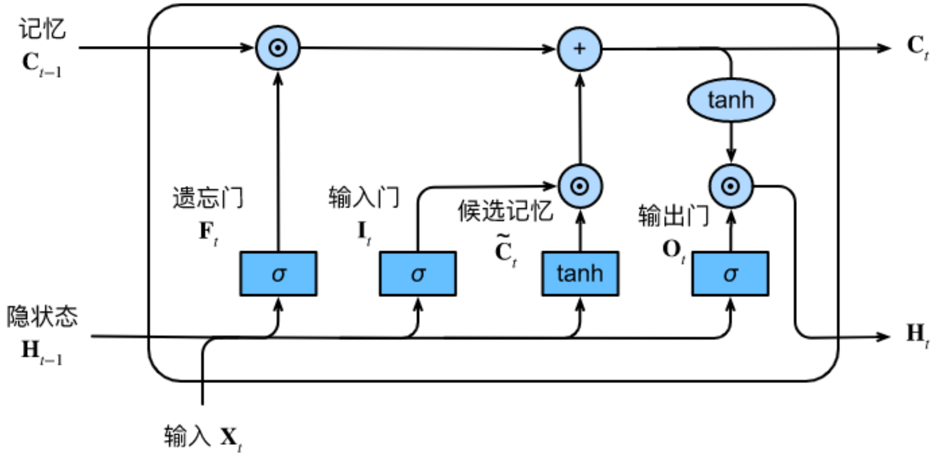
Training Process

Common Process of NN	My LSTM
for each epoch:	3
for each X, Y in data iter:	MyDataLoader
	2 隐藏状态初始化
1 正向传播：y_hat = net(X)	
4 损失计算：l = loss(y_hat, y)	
梯度归零：updater.zero_grad()	
梯度计算：l.backward()	5 梯度裁剪
参数更新：updater.step()	

y_hat = net(x)



- RNN model: Each neuro utilizes the information from the prior neuro



For each neuro,

- Inner structure is packaged neatly

```
self.layer = nn.LSTM(feature_size, hidden_size, num_layers)
self.linear = nn.Linear(hidden_size, output_size)
```

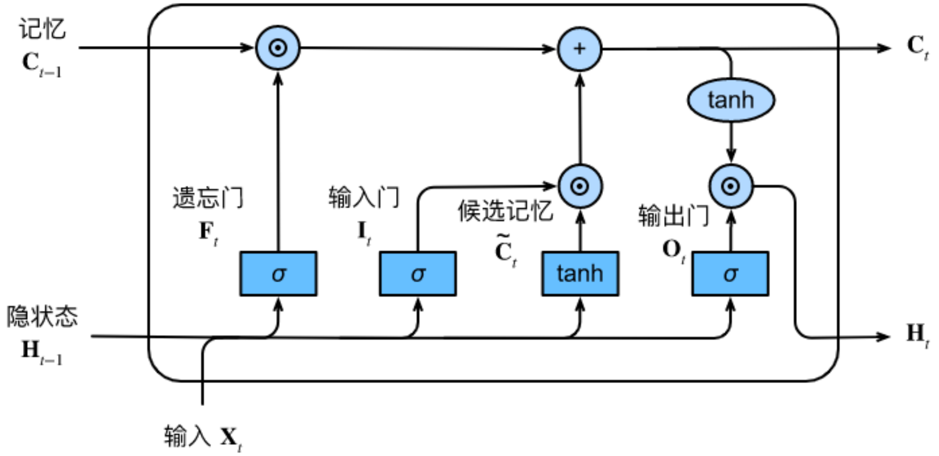
- What we need: C_0, H_0, X_t

```
output, state = self.layer(X, state)
y_hat = self.linear(output)
```

C_0, H_0

- Formula

$$\begin{cases} \mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i), \\ \mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f), \\ \mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o), \\ \tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c), \end{cases} \quad \begin{cases} \mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t. \\ \mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t). \\ \hat{Y}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q. \end{cases}$$



- Shape

W_x defined in nn.LSTM: [feature_size, hidden_size]
W_h defined in nn.LSTM: [hidden_size, hidden_size]
X_t: [batch_size, feature_size]
X: [time_steps, batch_size, feature_size]
W defined in nn.Linear: [hidden_size, output_size]
H_t, C_t: [batch_size, hidden_size]

- 2 隐藏状态初始化 C_0, H_0

```
(torch.zeros((num_layers,
              batch_size, hidden_size), device=device),
 torch.zeros((num_layers,
              batch_size, hidden_size), device=device))
```

❸ MyDataLoader – X_t

X: [time_steps, batch_size, feature_size]
- times_steps为超参数，表示每次预测要用过去多长时间的数据
- feature_size代表特征数，这是由因子数量决定的
- batch_size是可变的，这是小批量梯度下降中用到的样本数量

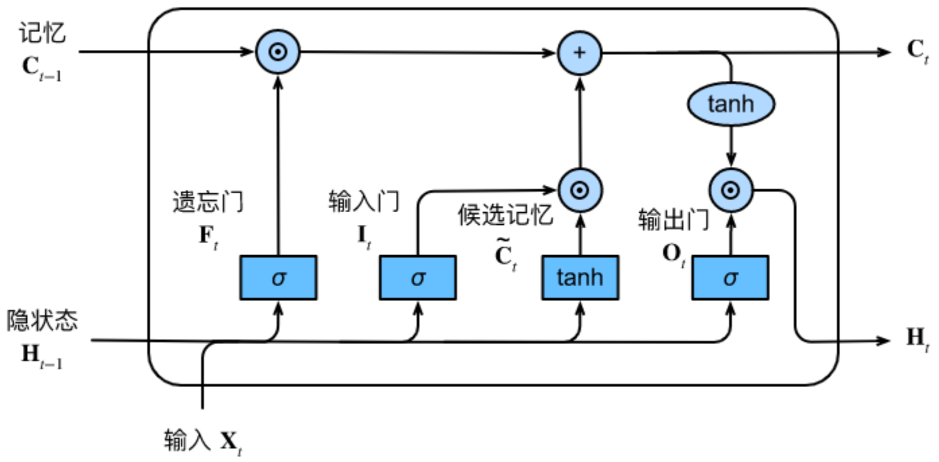
实现过程中有遇到很多问题——
假定当前有3000支股票，100期数据，300个因子，根据经验，time_steps=5是个不错的选择，现在需要根据当前数据生成X[5, batch_size, 300]
- Q1: X在时间上是连续的还是重叠的?
  - t1-t5, t6-t10, …, t96-t100 / t1-t5, t2-t6, t3-t7, …, t95-t99, t96-t100
- Q2: 在某些截面，一些股票会被禁止交易，如停牌、ST
  - batch_size可变 / batch_size不变，丢弃这些股票
  - 如果股票总数只有10，是否batch_size ≤10
- Q3: 在同一个X中，是否需要维持每只股票的位置一致?
  - 在每个time_step中，第一条样本为股票A，第二条样本为股票B，…

4 损失计算：l = loss(y_hat, y)

- Formula

$$\begin{cases} \mathbf{I}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i), \\ \mathbf{F}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f), \\ \mathbf{O}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o), \\ \tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c), \end{cases}$$

$$\begin{cases} \mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t. \\ \mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t). \\ \hat{Y}_t = \mathbf{H}_t\mathbf{W}_{hq} + \mathbf{b}_q. \end{cases}$$



- 只需要最后的$\hat{y}_T$，前$time_steps$的数据都是炮灰

```
y = (Y[-1].reshape(-1)).to(device)  # 最后一期y拉成向量
y_hat = y_hat[-1].reshape(-1)  # [批量大小,输出大小]
```

- 回归问题

```
loss = nn.MSELoss()
l = loss(y_hat, y.float())
```

- 分类问题

```
loss = nn.CrossEntropyLoss()
# 好像不需要 y_hat = nn.functional.softmax(y_hat, dim=1)
l = loss(y_hat, y.long())  # label一定要≥0
```

5

梯度裁剪

RNN模型的梯度爆炸和梯度消失问题：

- 在反向传播过程中产生长度为O(T)的矩阵乘法链
- 当T较大时，它可能导致数值不稳定
- RNN往往需要额外的方式来支持稳定训练

当梯度很大或者学习率很大时，都会出现模型反复跳跃，训练无法收敛的问题：

$$|f(\mathbf{x}) - f(\mathbf{x} - \eta\mathbf{g})| \leq L\eta\|\mathbf{g}\|$$

解决梯度爆炸问题：

- 限定一个小的学习率是可行的
- 但如果g的过大只是偶发性的 ➡ 梯度裁剪

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g}$$

```
@staticmethod
def grad_clipping(net, theta):
    # 所有层的参数统一裁剪
    params = [p for p in net.parameters() if p.requires_grad]
    norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in params))
    if norm > theta:
        for param in params:
            param.grad[:] *= theta / norm
```

Predict

X: [time_steps, batch_size, feature_size]

- times_steps为超参数，表示每次预测要用过去多长时间的数据
- feature_size代表特征数，这是由因子数量决定的
- batch_size是可变的，这是小批量梯度下降中用到的样本数量

假定当前有3000支股票，100期数据，300个因子，根据经验，time_steps=5是个不错的选择，现在需要根据当前数据生成X[5, batch_size, 300]

- 每次预测都重置隐藏状态
- 需要time_steps-1期的预热数据
- X在时间上是重叠的
  - t1-t5, t2-t6, t3-t7, …, t95-t99, t96-t100
- 如果某只股票不够连续的5期则不能预测
- batch_size可以随意变动
- 在一个X中，需要维持每只股票的位置一致

```
for inputs, _, days, secIDs in test_dataIter:
    batch_size = inputs.shape[1]
    state = net.begin_state(batch_size=batch_size, device=device)
    inputs = inputs.to(device)
    outputs, state = net(inputs, state)
    outputs = outputs.reshape((len(days), len(secIDs)))[-1]
```

谢谢大家！