lstm_apply

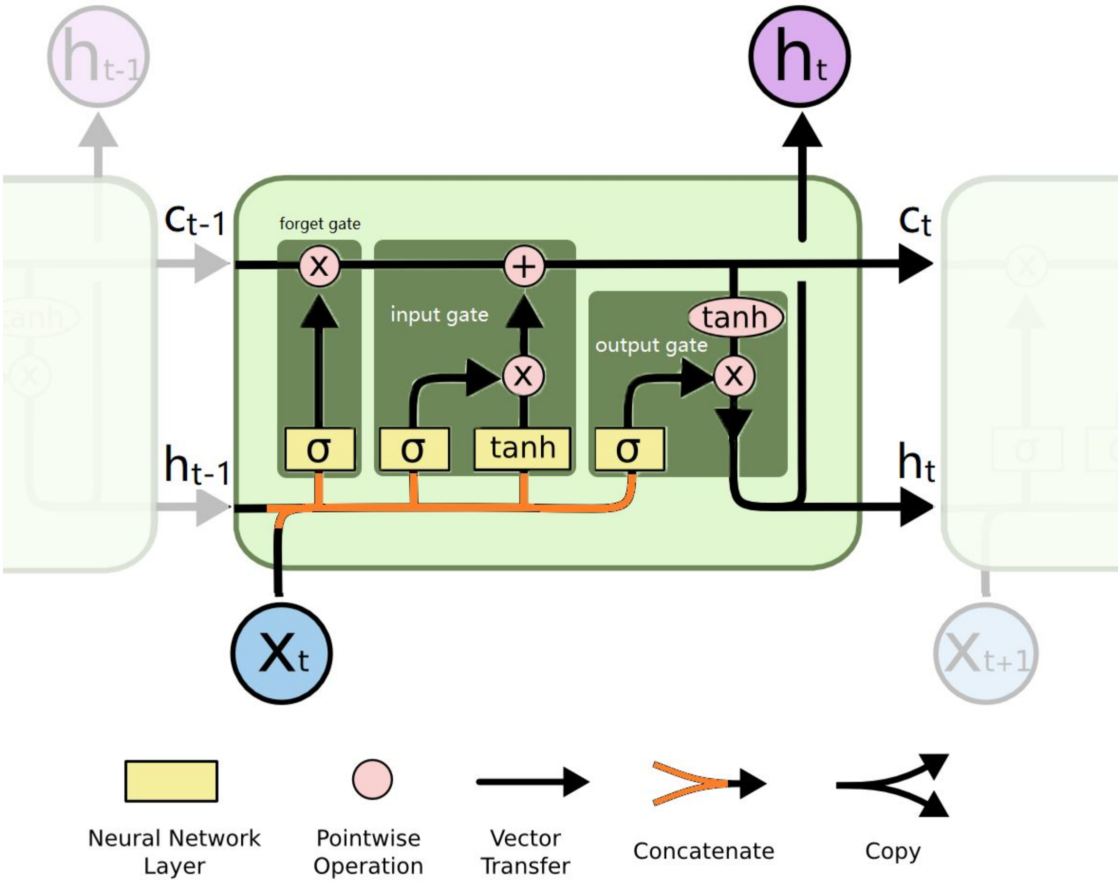
2023 年 2 月 18 日

1 lstm 的 demo

[1]:
```
import numpy as np
import torch
from torch import nn
import matplotlib.pyplot as plt
```

1.1 define the network

1.1.1 原理



$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

注：上面的下标是指从什么到什么，比如 W_{ii} 表示从输入到输入的权重，W_{ig} 表示从输入到门的权重，W_{hg} 表示从隐藏层到门的权重，W_{ho} 表示从隐藏层到输出的权重，W_{hf} 表示从隐藏层到遗忘门的权重，W_{hi} 表示从隐藏层到输入门的权重。b_{ii} 表示从输入到输入的偏置，b_{ig} 表示从输入到门的偏置，b_{hg} 表示从隐藏层到门的偏置，b_{ho} 表示从隐藏层到输出的偏置，b_{hf} 表示从隐藏层到遗忘门的偏置，b_{hi} 表示从隐藏层到输入门的偏置。i_t 表示输入门，f_t 表示遗忘门，g_t 表示门，o_t 表示输出门，c_t 表示记忆单元，h_t 表

示隐藏层。

[2]:
```python
class RegLSTM(nn.Module):
    def __init__(self, inp_dim, out_dim, mid_dim, mid_layers):
        super(RegLSTM, self).__init__()
        self.rnn = nn.LSTM(inp_dim, mid_dim, mid_layers)
        # rnn layer 在自然语言处理中，第一个参数通常是 embedding 的维度，第二个参数是隐藏层的维度，第三个参数是层数
        self.reg = nn.Sequential(
            nn.Linear(mid_dim, mid_dim),
            nn.Tanh(),
            nn.Linear(mid_dim, out_dim),
        )  # regression

    def forward(self, x):
        y = self.rnn(x)[0]  # y, (h, c) = self.rnn(x)

        seq_len, batch_size, hid_dim = y.shape
        y = y.view(-1, hid_dim) # y = y.view(seq_len * batch_size, hid_dim)
        y = self.reg(y)
        y = y.view(seq_len, batch_size, -1) # y = y.view(seq_len, batch_size,␣
↪out_dim)
        return y

    """
    PyCharm Crtl+click nn.LSTM() jump to code of PyTorch:
    Examples::
        >>> rnn = nn.LSTM(10, 20, 2)
        >>> input = torch.randn(5, 3, 10)
        >>> h0 = torch.randn(2, 3, 20)
        >>> c0 = torch.randn(2, 3, 20)
        >>> output, (hn, cn) = rnn(input, (h0, c0))
    """

    def output_y_hc_for_test(self, x, hc):
        # 后面计算 loss 的时候，需要用到 y 和 hc，所以这里需要单独写一个函数
        y, hc = self.rnn(x, hc)  # y, (h, c) = self.rnn(x)
```

```python
        seq_len, batch_size, hid_dim = y.size()
        y = y.view(-1, hid_dim)
        y = self.reg(y)
        y = y.view(seq_len, batch_size, -1)
        return y, hc
```

1.1.2 参数设定

[3]:
```python
inp_dim = 1 # 输入维度 我们是（reported result）一个维度
out_dim = 1 # 输出维度 我们是预测客流量，所以是 1
mid_dim = 10 # 隐藏层维度
mid_layers = 1 # 隐藏层层数
# batch_size = 12 * 4 # 我们划分成 48 个 batch <-- 后面改
batch_size = 100
mod_dir = '.'
```

1.1.3 load data

for data 1

[4]:
```python
import pandas as pd
# 读取数据
df = pd.read_csv('df_Number_of_reported_results.csv')
seq_number = df.values
# 需要先反转
seq_number = seq_number[::-1]
```

for data 2

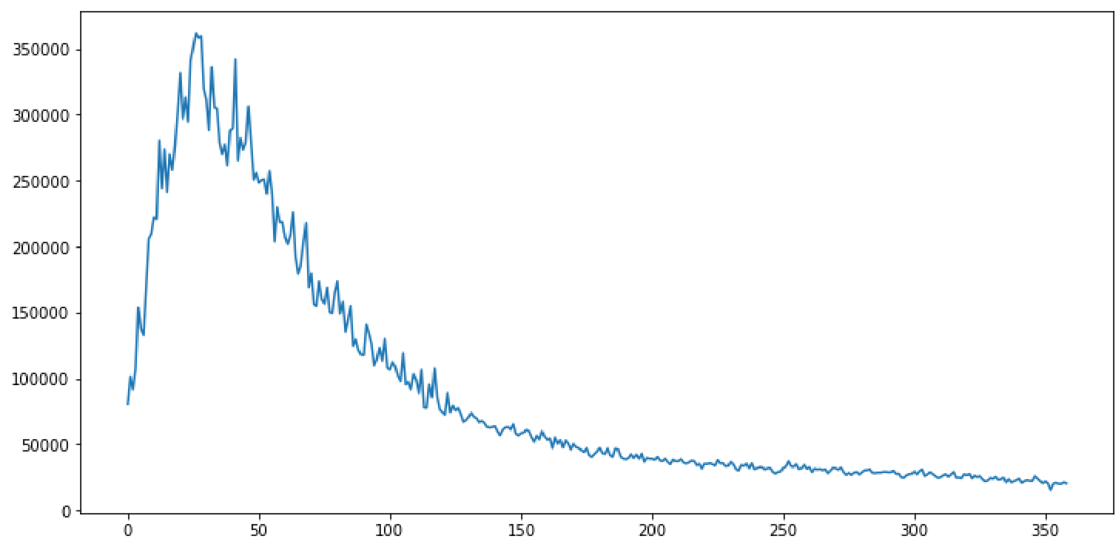
[5]:
```python
# # passengers number of international airline , 1949-01 ~ 1960-12 per month
# seq_number = np.array(
#     [112., 118., 132., 129., 121., 135., 148., 148., 136., 119., 104.,
#         118., 115., 126., 141., 135., 125., 149., 170., 170., 158., 133.,
#         114., 140., 145., 150., 178., 163., 172., 178., 199., 199., 184.,
#         162., 146., 166., 171., 180., 193., 181., 183., 218., 230., 242.,
#         209., 191., 172., 194., 196., 196., 236., 235., 229., 243., 264.,
#         272., 237., 211., 180., 201., 204., 188., 235., 227., 234., 264.,
#         302., 293., 259., 229., 203., 229., 242., 233., 267., 269., 270.,
```

```
#          315., 364., 347., 312., 274., 237., 278., 284., 277., 317., 313.,
#          318., 374., 413., 405., 355., 306., 271., 306., 315., 301., 356.,
#          348., 355., 422., 465., 467., 404., 347., 305., 336., 340., 318.,
#          362., 348., 363., 435., 491., 505., 404., 359., 310., 337., 360.,
#          342., 406., 396., 420., 472., 548., 559., 463., 407., 362., 405.,
#          417., 391., 419., 461., 472., 535., 622., 606., 508., 461., 390.,
#          432.], dtype=np.float32)
# # 给 seq_number 增加一个维度，变成 2 维的
# seq_number = seq_number[:, np.newaxis]
```

show data

[6]:
```
plt.figure(figsize=(12, 6))
plt.plot(seq_number)
```

[6]: [<matplotlib.lines.Line2D at 0x1d1a6108dc0>]



[7]:
```
seq_number.shape # 1 月 17 号到 12 月 31 号的数据，共 359 天
```

[7]: (359, 1)

[8]:
```
seq = seq_number # 所以最终的形式是（当天 reported result）
```

[9]:
```python
# 转化成浮点数
seq = seq.astype(np.float32)
seq[:5]
```

[9]:
```
array([[ 80630.],
       [101503.],
       [ 91477.],
       [107134.],
       [153880.]], dtype=float32)
```

[10]:
```python
# normalization
seq_mean = seq.mean(axis=0)
seq_std = seq.std(axis=0)
seq_norm = (seq - seq_mean) / seq_std
```

[11]:
```python
seq_norm[:5]
```

[11]:
```
array([[-0.11607783],
       [ 0.11818092],
       [ 0.00565861],
       [ 0.18137792],
       [ 0.7060107 ]], dtype=float32)
```

[12]:
```python
data = seq_norm
data_x = data[:-1, :] # 从 0 到倒数第二个
data_y = data[+1:, 0] # 从 1 到最后一个
assert data_x.shape[1] == inp_dim
print("data_x[:5]:", data_x[:5])
print("data_y[:5]:", data_y[:5])
print("data_x.shape:", data_x.shape)
print("data_y.shape:", data_y.shape)
```

```
data_x[:5]: [[-0.11607783]
 [ 0.11818092]
 [ 0.00565861]
 [ 0.18137792]
 [ 0.7060107 ]]
data_y[:5]: [0.11818092 0.00565861 0.18137792 0.7060107  0.5231423 ]
data_x.shape: (358, 1)
```

```
data_y.shape: (358,)
```

[13]:
```
## 2022.2.17 到 2022.10.29 都是训练集，2022.10.30 到 2022.12.31 都是测试集 也就是
一共 497 - 202 + 1 = 296 天
# 296/len(data_x) # <-- for data1
```

[14]:
```
# split train and test
# train_size = 296 # for data1
train_size = int(len(data_x) * 0.70) # 75% 的数据作为训练集 for data2

train_x = data_x[:train_size]
train_y = data_y[:train_size]

train_x = train_x.reshape((train_size, inp_dim)) # 296, 1
train_y = train_y.reshape((train_size, out_dim)) # 296, 1
print("train_x.shape: ", train_x.shape)
print("train_y.shape: ", train_y.shape)
```

```
train_x.shape:  (250, 1)
train_y.shape:  (250, 1)
```

1.1.4 build model

[15]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = RegLSTM(inp_dim, out_dim, mid_dim, mid_layers).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-2)
```

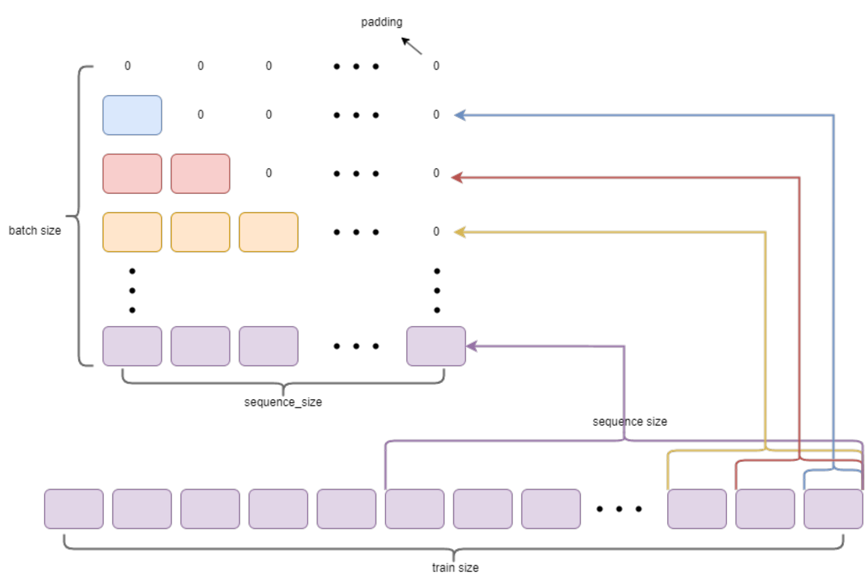
[16]:
```
from torchinfo import summary
summary(net, input_size=(batch_size - 1, batch_size, 1))
```

[16]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
RegLSTM                                  [99, 100, 1]              --
 LSTM: 1-1                               [99, 100, 10]             520
 Sequential: 1-2                         [9900, 1]                 --
```

```
    Linear: 2-1                       [9900, 10]           110
    Tanh: 2-2                         [9900, 10]           --
    Linear: 2-3                       [9900, 1]            11
================================================================================
==========
Total params: 641
Trainable params: 641
Non-trainable params: 0
Total mult-adds (M): 6.35
================================================================================
==========
Input size (MB): 0.04
Forward/backward pass size (MB): 1.66
Params size (MB): 0.00
Estimated Total Size (MB): 1.71
================================================================================
==========
```

1.1.5 train

制作 batch



我们首先制作 train 的 batch，然后训练模型。

制作 batch 的方法是选取不同开头但截止一样的序列，然后将这些序列组合成一个 batch。

制作一个 batch

[17]:
```python
# var_x = torch.tensor(train_x, dtype=torch.float32, device=device)
# var_y = torch.tensor(train_y, dtype=torch.float32, device=device)

# batch_var_x = list()
# batch_var_y = list()

# for roi_len in range(batch_size):
#     begin_idx = train_size - roi_len # train_size = 296
#     batch_var_x.append(var_x[begin_idx:])
#     batch_var_y.append(var_y[begin_idx:])
```

[18]:
```python
# print("var_x.shape: ", var_x.shape)
# print("var_y.shape: ", var_y.shape)
# print("batch_var_x[0].shape: ", batch_var_x[0].shape)
# print("batch_var_y[0].shape: ", batch_var_y[0].shape)
# print("batch_var_x[1].shape: ", batch_var_x[-1].shape)
# print("batch_var_y[1].shape: ", batch_var_y[-1].shape)
# print("batch_var_x.len: ", len(batch_var_x))
# print("batch_var_y.len: ", len(batch_var_y))
```

制作多个 mini_batch 使得一个 epoch 可以看完所有数据

[19]:
```python
batch_size
```

[19]: 100

[20]:
```python
train_size
```

[20]: 250

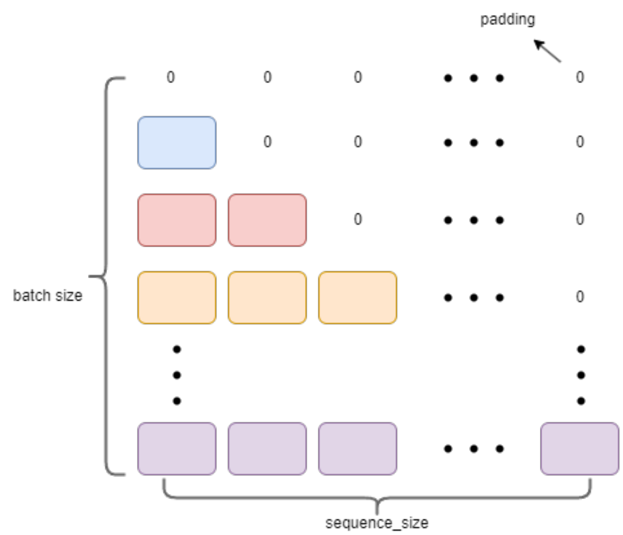
[21]:
```python
# 那么我们大概需要再制作 4 个 batch 就好了，重复是必须的
var_x = torch.tensor(train_x, dtype=torch.float32, device=device)
var_y = torch.tensor(train_y, dtype=torch.float32, device=device)

batch_var_x = list()
batch_var_y = list()

mini_batch_size = 5
```

```python
for num_of_batch in range(mini_batch_size):
    end_idx = np.random.randint(batch_size, train_size)
    for roi_len in range(batch_size):
        begin_idx = end_idx - roi_len
        batch_var_x.append(var_x[begin_idx:end_idx])
        batch_var_y.append(var_y[begin_idx:end_idx])
```

[22]:
```python
print("var_x.shape: ", var_x.shape)
print("var_y.shape: ", var_y.shape)
print("batch_var_x[0].shape: ", batch_var_x[0].shape)
print("batch_var_y[0].shape: ", batch_var_y[0].shape)
print("batch_var_x[-1].shape: ", batch_var_x[-1].shape)
print("batch_var_y[-1].shape: ", batch_var_y[-1].shape)
print("batch_var_x.len: ", len(batch_var_x))
print("batch_var_y.len: ", len(batch_var_y))
```

```
var_x.shape:  torch.Size([250, 1])
var_y.shape:  torch.Size([250, 1])
batch_var_x[0].shape:  torch.Size([0, 1])
batch_var_y[0].shape:  torch.Size([0, 1])
batch_var_x[-1].shape:  torch.Size([99, 1])
batch_var_y[-1].shape:  torch.Size([99, 1])
batch_var_x.len:  500
batch_var_y.len:  500
```

padding



我们看到不同的 batch 的 shape 是不一样的，但是我们需要的是一样的，所以我们需要对 batch 进行 padding，即在后面补 0，使得所有的 batch 的 shape 都一样

[23]:
```
from torch.nn.utils.rnn import pad_sequence
batch_var_x = pad_sequence(batch_var_x)
batch_var_y = pad_sequence(batch_var_y)
print("batch_var_x.shape: ", batch_var_x.shape)
print("batch_var_y.shape: ", batch_var_y.shape)
```

```
batch_var_x.shape:  torch.Size([99, 500, 1])
batch_var_y.shape:  torch.Size([99, 500, 1])
```

[24]:
```
batch_var_x[:5, 0, :] # 对应上图的蓝色圆角矩形上方第一行数据前五个
```

[24]:
```
tensor([[0.],
        [0.],
        [0.],
        [0.],
        [0.]])
```

[25]:
```
batch_var_x[:5, 1, :]  # 对应上图的蓝色圆角矩形所在行数据前五个
```

[25]:
```
tensor([[-0.0579],
        [ 0.0000],
        [ 0.0000],
        [ 0.0000],
        [ 0.0000]])
```

遗忘曲线控制 loss 这里为损失函数添加了类似遗忘曲线的东西，使得模型在训练的时候不会忘记之前的信息，而是会逐渐遗忘。

$$weight1_t = tanh(e * \frac{t}{len(train_y)}) \quad \text{where} \quad t \in [0, len(train_y))$$

试图改一下遗忘曲线，使得模型在训练的时候对近期记忆保持更多的记忆

$$weight2_t = tanh\left(\alpha * \left(\frac{t}{len(train_y)} - 1\right)\right) + 1 \quad \text{where} \quad t \in [0, len(train_y))$$

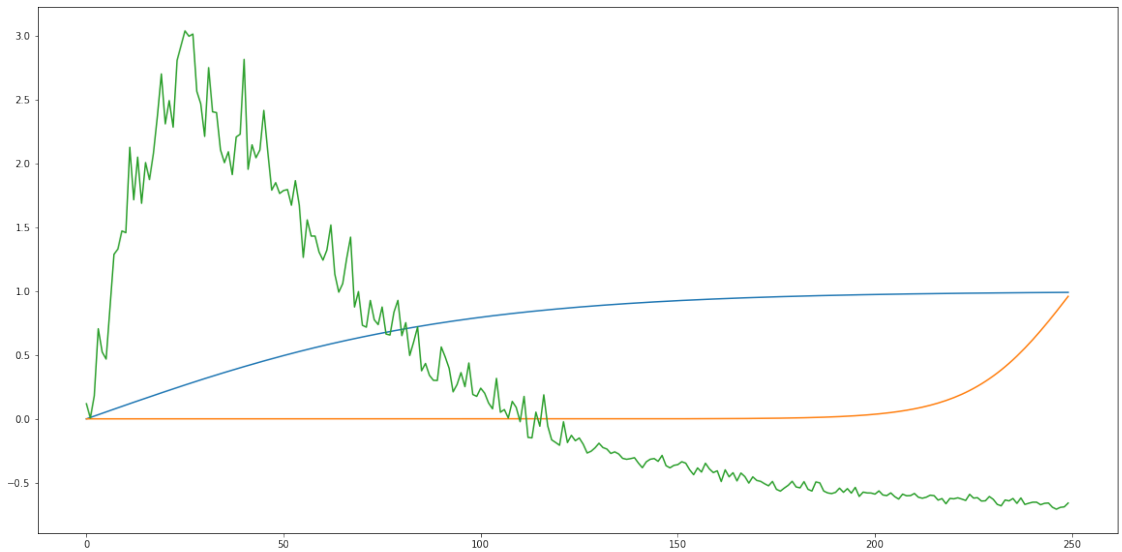
[26]:
```python
with torch.no_grad():
    weights = np.tanh(np.arange(len(train_y)) * (np.e / len(train_y)))
    weights = torch.tensor(weights, dtype=torch.float32, device=device)

with torch.no_grad():
    alpha = 10
    weights2 = np.tanh( alpha *(np.arange(len(train_y)) / len(train_y) - 1)) + 1
    weights2 = torch.tensor(weights2, dtype=torch.float32, device=device)
```

[27]:
```python
print("weights.shape: ", weights.shape)
```

```
weights.shape:  torch.Size([250])
```

[28]:
```python
# 画出 weights
plt.figure(figsize=(20, 10))
plt.plot(weights.cpu().numpy())
plt.plot(weights2.cpu().numpy())
# 画出客流量
plt.plot(train_y)
```

[28]:
```
[<matplotlib.lines.Line2D at 0x1d1a826f400>]
```



开始训练

[29]:
```
net.train()
print("Training Start")
for epoch in range(800):
    out = net(batch_var_x)

    # loss = (out - batch_var_y) ** 2 * weights # <-- 使用 weights
    # test mse loss:  32856874.47789988
    # loss = (out - batch_var_y) ** 2 * weights2 # <-- 使用 weights2
    # alpha = 2 test mse loss:  37580656.65421245
    # alpha = 5 test mse loss:  28990949.182173382
    # alpha = 7 test mse loss:  24158278.41746032
    # alpha = 10 test mse loss:  32370050.34371184
    loss = (out - batch_var_y) ** 2 # <-- 不使用 weights
    # test mse loss:  13283323.04859585 结果最好

    loss = loss.mean()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
    if epoch % 50 == 0:
        print('Epoch: {:4}, Loss: {:.5f}'.format(epoch, loss.item()))
print("Training End, Loss: {:.5f}".format(loss.item()))
```

```
Training Start
Epoch:    0, Loss: 0.37251
Epoch:   50, Loss: 0.01125
Epoch:  100, Loss: 0.00961
Epoch:  150, Loss: 0.00889
Epoch:  200, Loss: 0.00834
Epoch:  250, Loss: 0.00785
Epoch:  300, Loss: 0.00748
Epoch:  350, Loss: 0.00717
Epoch:  400, Loss: 0.00699
Epoch:  450, Loss: 0.00691
Epoch:  500, Loss: 0.00680
Epoch:  550, Loss: 0.00672
Epoch:  600, Loss: 0.00664
Epoch:  650, Loss: 0.00657
Epoch:  700, Loss: 0.00685
Epoch:  750, Loss: 0.00643
Training End, Loss: 0.00636
```

保存参数

[30]:
```
# torch.save(net.state_dict(), '{}/net.pth'.format(mod_dir))
# print("Save in:", '{}/net.pth'.format(mod_dir))
```

1.1.6 eval

制作 test set

[31]:
```
test_x = seq_norm.copy()
test_x[train_size:, 0] = 0 # 设置为 0, 表示预测
test_x = test_x[:, np.newaxis, :]
test_x = torch.tensor(test_x, dtype=torch.float32, device=device)
print("test_x.shape: ", test_x.shape)
```

```
test_x.shape:  torch.Size([359, 1, 1])
```

[32]:
```
# net.load_state_dict(torch.load('{}/net.pth'.format(mod_dir),␣
↪map_location=lambda storage, loc: storage))
net.eval()
```

[32]:
```
RegLSTM(
  (rnn): LSTM(1, 10)
  (reg): Sequential(
    (0): Linear(in_features=10, out_features=10, bias=True)
    (1): Tanh()
    (2): Linear(in_features=10, out_features=1, bias=True)
  )
)
```

[33]:
```
with torch.no_grad():
    '''simple way is elegant'''
    for i in range(train_size, len(data)):
        test_y = net(test_x[:i])
        test_x[i, 0, 0] = test_y[-1]

    pred_y = test_x[1:, 0, 0]
    pred_y = pred_y.cpu().data.numpy()

    diff_y = pred_y[train_size:] - data_y[train_size:]
    l2_loss = np.mean(diff_y ** 2)
    print(" test mse loss: ", l2_loss)
```

```
 test mse loss:  0.007266506
```

for data 1

[34]:
```
# origin_data = df.values
# # 反转
# origin_data = origin_data[::-1]
```

for data 2

[35]:
```
origin_data = seq.copy()
```

[36]:
```
print("pred_y.shape: ", pred_y.shape)
print("data_y.shape: ", data_y.shape)
print("the last pred_y: ", pred_y[-1])
print("the last data_y: ", data_y[-1])
```

```
pred_y.shape:  (358,)
data_y.shape:  (358,)
the last pred_y:  -0.646692
the last data_y:  -0.79226667
```

数据还原

[37]:
```
mean = origin_data.mean()
std = origin_data.std()
print("mean: ", mean)
print("std: ", std)

# 将预测值还原
pred_y = pred_y * std + mean
# 设置成 int 类型
pred_y = pred_y.astype(int)
```

```
mean:  90972.805
std:  89102.33
```

求 MSE (Mean Squared Error)

[38]:
```
# 和真实值对比求误差
diff_y = pred_y[train_size:] - origin_data[train_size:]
l2_loss = np.mean(diff_y ** 2)
print(" test mse loss: ", l2_loss)
```

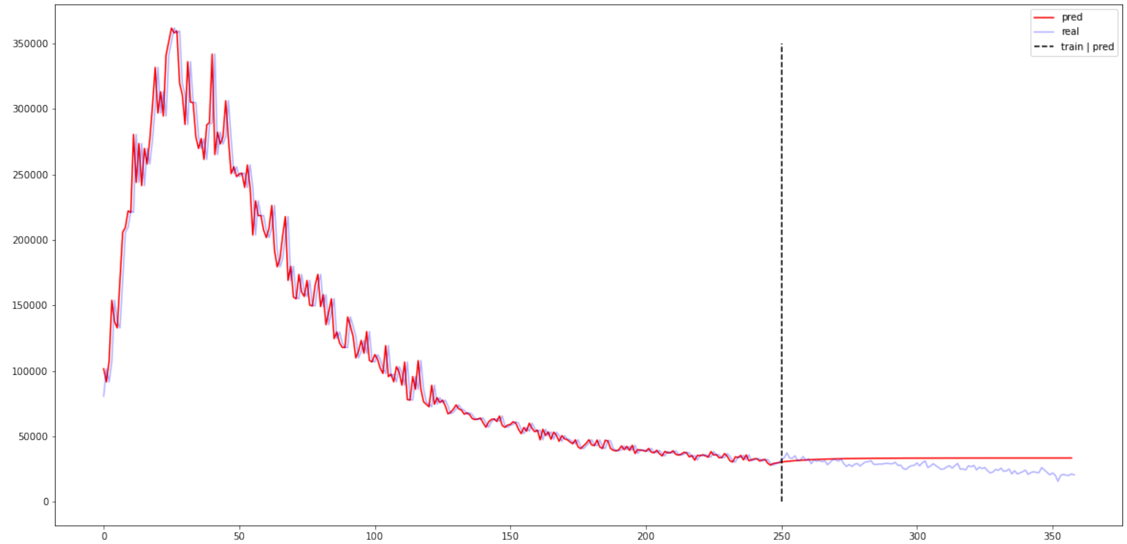
```
 test mse loss:  53422339.48335032
```

预测曲线图

[39]:
```
plt.figure(figsize=(20,10))
plt.plot(pred_y, 'r', label='pred')
plt.plot(origin_data, 'b', label='real', alpha=0.3)
plt.plot([train_size, train_size], [0, 350000], color='k', label='train |
 ↪pred', linestyle='--')
```

```
plt.legend(loc='best')
# plt.savefig('pred_with_regularization_batch_lstm1.png')
```

[39]: <matplotlib.legend.Legend at 0x1d1a82f2250>



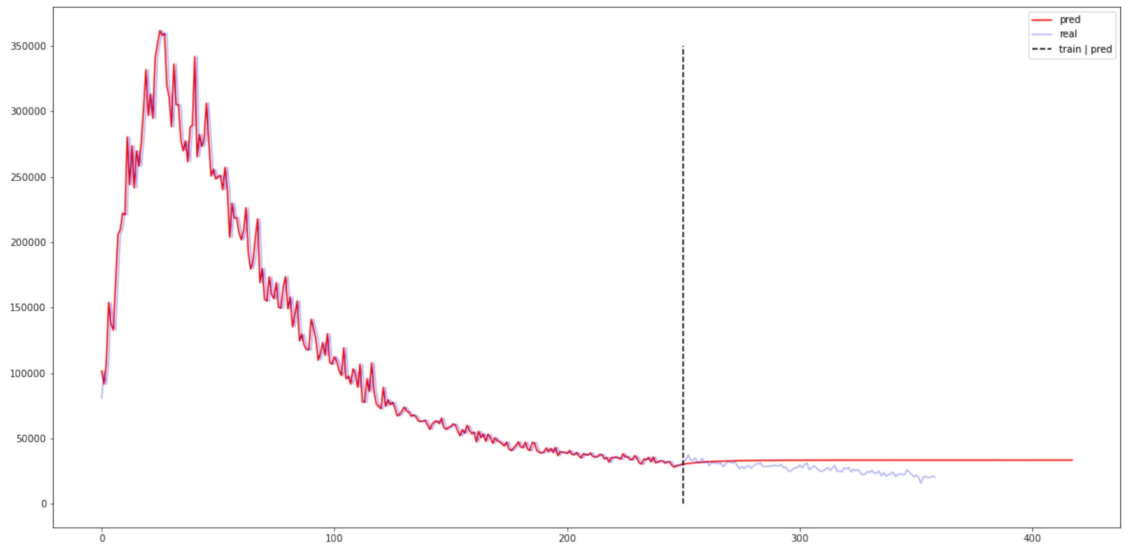
长期预测

[40]:
```
pred_result = seq_norm.copy()
pred_result = np.concatenate((pred_result, np.zeros((60, 1))), axis=0)
pred_result[train_size:, 0] = 0 # 设置为 0, 表示预测
pred_result = pred_result[:, np.newaxis, :]
pred_result = torch.tensor(pred_result, dtype=torch.float32, device=device)

with torch.no_grad():
    '''simple way is elegant'''
    for i in range(train_size, len(data)+60):
        test_y = net(pred_result[:i])
        pred_result[i, 0, 0] = test_y[-1]

    pred_y = pred_result[1:, 0, 0]
    pred_y = pred_y.cpu().data.numpy()
```

[41]:
```
# 还原
pred_y = pred_y * std + mean
pred_y = pred_y.astype(int)
```

[42]:
```
plt.figure(figsize=(20,10))
plt.plot(pred_y, 'r', label='pred')
plt.plot(origin_data, 'b', label='real', alpha=0.3)
plt.plot([train_size, train_size], [0, 350000], color='k', label='train |␣
↪pred', linestyle='--')
plt.legend(loc='best')
# plt.savefig('pred_with_weight2_alpha_' + str(alpha) + '.png')
# plt.savefig('pred_with_no_weight_batchsize_' + str(batch_size) + '.png')
```

[42]:
```
<matplotlib.legend.Legend at 0x1d1a9383940>
```



[43]:
```
print("pred_y.shape: ", pred_y.shape)
print("final pred_y: ", pred_y[-1])
```

```
pred_y.shape:  (418,)
final pred_y:  33352
```

1.1.7 总计改进

由上面可以发现不加 weights 训练效果是最好的，我们多次进行不加 weights 的训练查看效果。

我们下一步可以做的工作有，- 制作 mini_batch，训练更多的数据而不是只有我们训练的一个 batch 而已 - 重新确定好训练集和测试集的比例分割数据

最优的一幅图