This part of the note is based on:

https://medium.com/analytics-vidhya/linear-regression-with-pytorch-147fed55f138

https://medium.com/analytics-vidhya/linear-regression-with-pytorch-147fed55f138

https://blog.csdn.net/zw__chen/article/details/82806900

https://blog.csdn.net/qq_40211493/article/details/107529148

https://online.stat.psu.edu/stat462/node/132/

https://www.cnblogs.com/ranjiewen/p/10128046.html

We need to figure out how to apply linear regression with the help of pytorch.

本文旨在使用 pytorch 实现线性回归。

The first thing we need to understand is what linear regression is. We had worked on similar task during middle school. If two points are given, we could find the line go through these two points. Linear regression is to find the best fit line of a set of points (these points are known as data).

For simple case: $y = kx + b$, we can rewrite it into (add error part ε):

$$y = \beta_0 + \beta_1 x + \varepsilon$$

If we extent it to n equations:

$$y_1 = \beta_0 + \beta_1 x_1 + \varepsilon_1$$

$$y_2 = \beta_0 + \beta_1 x_2 + \varepsilon_2$$

$$\dots\dots$$

$$y_n = \beta_0 + \beta_1 x_n + \varepsilon_n$$

If put those equations in matrixes:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

$$\boldsymbol{Y} = \boldsymbol{X}\beta + \varepsilon$$

$\boldsymbol{X}$ is an $n \times 2$ matrix, containing feature data.

$\boldsymbol{Y}$ is an $n \times 1$ matrix(vector), containing target (what the result)

β is an 2×1 matrix(vector), what we used to represent the polynomial: $y = kx + b$ (k and b)

ε is an $n \times 1$ matrix(vector), error.

What we need to do to create the model:

Step 1:

Import libraries and load data set (example here create dataset by the numpy.array() function, it is used to create tensors, we can find the method to create tensors in Note 1). The dataset will be tensor.

Example:

```
import torch
import torch.nn as nn
import pandas as pd
import numpy as np
# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43], [91, 88, 64], [87, 134, 58],
                   [102, 43, 37], [69, 96, 70], [73, 67, 43],
                   [91, 88, 64], [87, 134, 58], [102, 43, 37],
                   [69, 96, 70], [73, 67, 43], [91, 88, 64],
                   [87, 134, 58], [102, 43, 37], [69, 96, 70]],
                  dtype='float32')
# Targets (apples, oranges)
targets = np.array([[56, 70], [81, 101], [119, 133],
                    [22, 37], [103, 119], [56, 70],
                    [81, 101], [119, 133], [22, 37],
                    [103, 119], [56, 70], [81, 101],
                    [119, 133], [22, 37], [103, 119]],
                   dtype='float32')
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

print(inputs)
print(targets)
```

output:

```
tensor([[ 73.,  67.,  43.],
        [ 91.,  88.,  64.],
        [ 87., 134.,  58.],
        [102.,  43.,  37.],
        [ 69.,  96.,  70.],
        [ 73.,  67.,  43.],
```

```
        [ 91.,  88.,  64.],
        [ 87., 134.,  58.],
        [102.,  43.,  37.],
        [ 69.,  96.,  70.],
        [ 73.,  67.,  43.],
        [ 91.,  88.,  64.],
        [ 87., 134.,  58.],
        [102.,  43.,  37.],
        [ 69.,  96.,  70.]])
tensor([[ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
        [103., 119.],
        [ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
        [103., 119.],
        [ 56.,  70.],
        [ 81., 101.],
        [119., 133.],
        [ 22.,  37.],
             [103., 119.]])
```

Step 2:

To combine the data we create/import before into a dataset, and split them into branches.

Example:

```
# Tensor Data set
from torch.utils.data import TensorDataset
# Define dataset
train_ds = TensorDataset(inputs, targets)

#Data loader
from torch.utils.data import DataLoader
# Define data loader
batch_size = 5
train_dl = DataLoader(train_ds, batch_size, shuffle=True)
print(type(train_ds))
print(train_ds[0:3])
print(type(train_dl))
"""
TensorDataset(): used to zip the tensors (input and target)
Dataloader(): used to break data into branches

"""
```

output:

```
<class 'torch.utils.data.dataset.TensorDataset'>
(tensor([[ 73.,  67.,  43.],
        [ 91.,  88.,  64.],
        [ 87., 134.,  58.]]), tensor([[ 56.,  70.],
        [ 81., 101.],
        [119., 133.]]))
<class 'torch.utils.data.dataloader.DataLoader'>
'
\nDataloader():\n\n
```

Step 3:

Define model, loss function and optimizer

Example:

```
# Define linear model
"""
variables in nn.Linear:
 (in_features, out_features, bias = True)
 bias has a default:True, means this module has bias
 The dataset has 3 input features, and 2 outputs
"""
model = nn.Linear(3, 2)
print(model.weight)
print(model.bias)
# Parameters
list(model.parameters())
# Define Loss
import torch.nn.functional as F
loss_fn = F.mse_loss
# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)
#lr for learning rate

print('model, loss function and optimizer')
print(model)
print(loss_fn)
print(opt)
print(train_dl.prefetch_factor)#default 2
```

output:

```
Parameter containing:
tensor([[ 0.4174,  0.2530, -0.0019],
        [ 0.1426, -0.4794, -0.0903]], requires_grad=True)
Parameter containing:
tensor([-0.0875,  0.1758], requires_grad=True)
```

```
model, loss function and optimizer
Linear(in_features=3, out_features=2, bias=True)
<function mse_loss at 0x7f3b27989ef0>
SGD (
Parameter Group 0
    dampening: 0
    lr: 1e-05
    momentum: 0
    nesterov: False
    weight_decay: 0
)
2
```

Step 4:

Train & fit model , generate predictions

Example:

```
# Utility function to train the model
def fit(num_epochs, model, loss_fn, opt, train_dl):

    # Repeat for given number of epochs
    for epoch in range(num_epochs):

        # Train with batches of data
        for xb,yb in train_dl:

            # 1. Generate predictions
            pred = model(xb)

            # 2. Calculate loss
            loss = loss_fn(pred, yb)

            # 3. Compute gradients
            loss.backward()

            # 4. Update parameters using gradients
            opt.step()

            # 5. Reset the gradients to zero
            opt.zero_grad()

        # Print the progress
        if (epoch+1) % 10 == 0:
            print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs
, loss.item()))
```

```
#fit model for 100 epochs
fit(100,model,loss_fn,opt,train_dl)
# Generate predictions
preds = model(inputs)

print(preds)
```

output:

```
Parameter containing:
tensor([[-0.3951, -0.2517,  0.1787],
        [ 0.3952,  0.4714, -0.2303]], requires_grad=True)
Parameter containing:
tensor([0.3579, 0.2132], requires_grad=True)
Epoch [10/100], Loss: 356.7490
Epoch [20/100], Loss: 237.5122
Epoch [30/100], Loss: 168.8583
Epoch [40/100], Loss: 29.0133
Epoch [50/100], Loss: 88.0375
Epoch [60/100], Loss: 85.5572
Epoch [70/100], Loss: 90.6686
Epoch [80/100], Loss: 67.7489
Epoch [90/100], Loss: 67.5613
Epoch [100/100], Loss: 32.4097
tensor([[ 58.1472,  71.6851],
        [ 82.4975,  95.5350],
        [116.4632, 140.8147],
        [ 26.3940,  45.4751],
        [ 99.3963, 105.4259],
        [ 58.1472,  71.6851],
        [ 82.4975,  95.5350],
        [116.4632, 140.8147],
        [ 26.3940,  45.4751],
        [ 99.3963, 105.4259],
        [ 58.1472,  71.6851],
        [ 82.4975,  95.5350],
        [116.4632, 140.8147],
        [ 26.3940,  45.4751],
        [ 99.3963, 105.4259]], grad_fn=<AddmmBackward0>)
```

So the code will be like:

```
import torch
import torch.nn as nn
import pandas as pd
import numpy as np
# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43], [91, 88, 64], [87, 134, 58],
                   [102, 43, 37], [69, 96, 70], [73, 67, 43],
```

```
                    [91, 88, 64], [87, 134, 58], [102, 43, 37],
                    [69, 96, 70], [73, 67, 43], [91, 88, 64],
                    [87, 134, 58], [102, 43, 37], [69, 96, 70]],
                   dtype='float32')
# Targets (apples, oranges)
targets = np.array([[56, 70], [81, 101], [119, 133],
                    [22, 37], [103, 119], [56, 70],
                    [81, 101], [119, 133], [22, 37],
                    [103, 119], [56, 70], [81, 101],
                    [119, 133], [22, 37], [103, 119]],
                   dtype='float32')
inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

# print(inputs)
# print(targets)

# Tensor Data set
from torch.utils.data import TensorDataset
# Define dataset
train_ds = TensorDataset(inputs, targets)

#Data loader
from torch.utils.data import DataLoader
# Define data loader
batch_size = 5
train_dl = DataLoader(train_ds, batch_size, shuffle=True)

# Define linear model
"""
variables in nn.Linear:
 (in_features, out_features, bias = True)
 bias has a default:True, means this module has bias
 The dataset has 3 input features, and 2 outputs
"""
model = nn.Linear(3, 2)
print(model.weight)
print(model.bias)
# Parameters
list(model.parameters())
# Define Loss
import torch.nn.functional as F
loss_fn = F.mse_loss
# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)
```

```
#lr for learning rate

# Utility function to train the model
def fit(num_epochs, model, loss_fn, opt, train_dl):

    # Repeat for given number of epochs
    for epoch in range(num_epochs):

        # Train with batches of data
        for xb,yb in train_dl:

            # 1. Generate predictions
            pred = model(xb)

            # 2. Calculate loss
            loss = loss_fn(pred, yb)

            # 3. Compute gradients
            loss.backward()

            # 4. Update parameters using gradients
            opt.step()

            # 5. Reset the gradients to zero
            opt.zero_grad()

        # Print the progress
        if (epoch+1) % 10 == 0:
            print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs
, loss.item()))
#fit model for 100 epochs
fit(100,model,loss_fn,opt,train_dl)
# Generate predictions
preds = model(inputs)

print(preds)
```

Another example

```
import torch
import torch.nn as nn
import pandas as pd
```

```
import numpy as np
from sklearn import datasets

#Iris dataset was used here
iris=datasets.load_iris()
# Input
inputs = iris['data']
# Targets
targets = iris['target']

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)

print(inputs)
print(targets)

# Tensor Data set
from torch.utils.data import TensorDataset
# Define dataset
train_ds = TensorDataset(inputs, targets.to(torch.float32))
#tensor = tensor.to(torch.float32)
#Data loader
from torch.utils.data import DataLoader
# Define data loader
batch_size = 5
train_dl = DataLoader(train_ds, batch_size, shuffle=True)

# Define linear model
"""
variables in nn.Linear:
 (in_features, out_features, bias = True)
 bias has a default:True, means this module has bias
 The dataset has 3 input features, and 2 outputs
"""
model = nn.Linear(4,1)
print(model.weight)
print(model.bias)
# Parameters
list(model.parameters())
# Define Loss
import torch.nn.functional as F
loss_fn = F.mse_loss
# Define optimizer
opt = torch.optim.SGD(model.parameters(), lr=1e-5)
#lr for learning rate
```

```
# Utility function to train the model
def fit(num_epochs, model, loss_fn, opt, train_dl):

    # Repeat for given number of epochs
    for epoch in range(num_epochs):

        # Train with batches of data
        for xb,yb in train_dl:

            # 1. Generate predictions
            pred = model(xb.to(torch.float32))

            # 2. Calculate loss
            loss = loss_fn(pred, yb)

            # 3. Compute gradients
            loss.backward()

            # 4. Update parameters using gradients
            opt.step()

            # 5. Reset the gradients to zero
            opt.zero_grad()

        # Print the progress
        if (epoch+1) % 10 == 0:
            print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs
, loss.item()))
#fit model for 100 epochs
fit(100,model,loss_fn,opt,train_dl)
# Generate predictions
preds = model(inputs.to(torch.float32))

print(preds)
```

output:

```
tensor([[5.1000, 3.5000, 1.4000, 0.2000],
    [4.9000, 3.0000, 1.4000, 0.2000],
    [4.7000, 3.2000, 1.3000, 0.2000],
    [4.6000, 3.1000, 1.5000, 0.2000],
    [5.0000, 3.6000, 1.4000, 0.2000],
    [5.4000, 3.9000, 1.7000, 0.4000],
```

[4.6000, 3.4000, 1.4000, 0.3000],
[5.0000, 3.4000, 1.5000, 0.2000],
[4.4000, 2.9000, 1.4000, 0.2000],
[4.9000, 3.1000, 1.5000, 0.1000],
[5.4000, 3.7000, 1.5000, 0.2000],
[4.8000, 3.4000, 1.6000, 0.2000],
[4.8000, 3.0000, 1.4000, 0.1000],
[4.3000, 3.0000, 1.1000, 0.1000],
[5.8000, 4.0000, 1.2000, 0.2000],
[5.7000, 4.4000, 1.5000, 0.4000],
[5.4000, 3.9000, 1.3000, 0.4000],
[5.1000, 3.5000, 1.4000, 0.3000],
[5.7000, 3.8000, 1.7000, 0.3000],
[5.1000, 3.8000, 1.5000, 0.3000],
[5.4000, 3.4000, 1.7000, 0.2000],
[5.1000, 3.7000, 1.5000, 0.4000],
[4.6000, 3.6000, 1.0000, 0.2000],
[5.1000, 3.3000, 1.7000, 0.5000],
[4.8000, 3.4000, 1.9000, 0.2000],
[5.0000, 3.0000, 1.6000, 0.2000],
[5.0000, 3.4000, 1.6000, 0.4000],
[5.2000, 3.5000, 1.5000, 0.2000],
[5.2000, 3.4000, 1.4000, 0.2000],
[4.7000, 3.2000, 1.6000, 0.2000],
[4.8000, 3.1000, 1.6000, 0.2000],
[5.4000, 3.4000, 1.5000, 0.4000],
[5.2000, 4.1000, 1.5000, 0.1000],
[5.5000, 4.2000, 1.4000, 0.2000],
[4.9000, 3.1000, 1.5000, 0.2000],
[5.0000, 3.2000, 1.2000, 0.2000],
[5.5000, 3.5000, 1.3000, 0.2000],
[4.9000, 3.6000, 1.4000, 0.1000],
[4.4000, 3.0000, 1.3000, 0.2000],
[5.1000, 3.4000, 1.5000, 0.2000],
[5.0000, 3.5000, 1.3000, 0.3000],
[4.5000, 2.3000, 1.3000, 0.3000],
[4.4000, 3.2000, 1.3000, 0.2000],
[5.0000, 3.5000, 1.6000, 0.6000],
[5.1000, 3.8000, 1.9000, 0.4000],
[4.8000, 3.0000, 1.4000, 0.3000],
[5.1000, 3.8000, 1.6000, 0.2000],
[4.6000, 3.2000, 1.4000, 0.2000],
[5.3000, 3.7000, 1.5000, 0.2000],
[5.0000, 3.3000, 1.4000, 0.2000],
[7.0000, 3.2000, 4.7000, 1.4000],
[6.4000, 3.2000, 4.5000, 1.5000],
[6.9000, 3.1000, 4.9000, 1.5000],
[5.5000, 2.3000, 4.0000, 1.3000],
[6.5000, 2.8000, 4.6000, 1.5000],
[5.7000, 2.8000, 4.5000, 1.3000],
[6.3000, 3.3000, 4.7000, 1.6000],
[4.9000, 2.4000, 3.3000, 1.0000],
[6.6000, 2.9000, 4.6000, 1.3000],
[5.2000, 2.7000, 3.9000, 1.4000],
[5.0000, 2.0000, 3.5000, 1.0000],

[5.9000, 3.0000, 4.2000, 1.5000],
[6.0000, 2.2000, 4.0000, 1.0000],
[6.1000, 2.9000, 4.7000, 1.4000],
[5.6000, 2.9000, 3.6000, 1.3000],
[6.7000, 3.1000, 4.4000, 1.4000],
[5.6000, 3.0000, 4.5000, 1.5000],
[5.8000, 2.7000, 4.1000, 1.0000],
[6.2000, 2.2000, 4.5000, 1.5000],
[5.6000, 2.5000, 3.9000, 1.1000],
[5.9000, 3.2000, 4.8000, 1.8000],
[6.1000, 2.8000, 4.0000, 1.3000],
[6.3000, 2.5000, 4.9000, 1.5000],
[6.1000, 2.8000, 4.7000, 1.2000],
[6.4000, 2.9000, 4.3000, 1.3000],
[6.6000, 3.0000, 4.4000, 1.4000],
[6.8000, 2.8000, 4.8000, 1.4000],
[6.7000, 3.0000, 5.0000, 1.7000],
[6.0000, 2.9000, 4.5000, 1.5000],
[5.7000, 2.6000, 3.5000, 1.0000],
[5.5000, 2.4000, 3.8000, 1.1000],
[5.5000, 2.4000, 3.7000, 1.0000],
[5.8000, 2.7000, 3.9000, 1.2000],
[6.0000, 2.7000, 5.1000, 1.6000],
[5.4000, 3.0000, 4.5000, 1.5000],
[6.0000, 3.4000, 4.5000, 1.6000],
[6.7000, 3.1000, 4.7000, 1.5000],
[6.3000, 2.3000, 4.4000, 1.3000],
[5.6000, 3.0000, 4.1000, 1.3000],
[5.5000, 2.5000, 4.0000, 1.3000],
[5.5000, 2.6000, 4.4000, 1.2000],
[6.1000, 3.0000, 4.6000, 1.4000],
[5.8000, 2.6000, 4.0000, 1.2000],
[5.0000, 2.3000, 3.3000, 1.0000],
[5.6000, 2.7000, 4.2000, 1.3000],
[5.7000, 3.0000, 4.2000, 1.2000],
[5.7000, 2.9000, 4.2000, 1.3000],
[6.2000, 2.9000, 4.3000, 1.3000],
[5.1000, 2.5000, 3.0000, 1.1000],
[5.7000, 2.8000, 4.1000, 1.3000],
[6.3000, 3.3000, 6.0000, 2.5000],
[5.8000, 2.7000, 5.1000, 1.9000],
[7.1000, 3.0000, 5.9000, 2.1000],
[6.3000, 2.9000, 5.6000, 1.8000],
[6.5000, 3.0000, 5.8000, 2.2000],
[7.6000, 3.0000, 6.6000, 2.1000],
[4.9000, 2.5000, 4.5000, 1.7000],
[7.3000, 2.9000, 6.3000, 1.8000],
[6.7000, 2.5000, 5.8000, 1.8000],
[7.2000, 3.6000, 6.1000, 2.5000],
[6.5000, 3.2000, 5.1000, 2.0000],
[6.4000, 2.7000, 5.3000, 1.9000],
[6.8000, 3.0000, 5.5000, 2.1000],
[5.7000, 2.5000, 5.0000, 2.0000],
[5.8000, 2.8000, 5.1000, 2.4000],
[6.4000, 3.2000, 5.3000, 2.3000],

```
        [6.5000, 3.0000, 5.5000, 1.8000],
        [7.7000, 3.8000, 6.7000, 2.2000],
        [7.7000, 2.6000, 6.9000, 2.3000],
        [6.0000, 2.2000, 5.0000, 1.5000],
        [6.9000, 3.2000, 5.7000, 2.3000],
        [5.6000, 2.8000, 4.9000, 2.0000],
        [7.7000, 2.8000, 6.7000, 2.0000],
        [6.3000, 2.7000, 4.9000, 1.8000],
        [6.7000, 3.3000, 5.7000, 2.1000],
        [7.2000, 3.2000, 6.0000, 1.8000],
        [6.2000, 2.8000, 4.8000, 1.8000],
        [6.1000, 3.0000, 4.9000, 1.8000],
        [6.4000, 2.8000, 5.6000, 2.1000],
        [7.2000, 3.0000, 5.8000, 1.6000],
        [7.4000, 2.8000, 6.1000, 1.9000],
        [7.9000, 3.8000, 6.4000, 2.0000],
        [6.4000, 2.8000, 5.6000, 2.2000],
        [6.3000, 2.8000, 5.1000, 1.5000],
        [6.1000, 2.6000, 5.6000, 1.4000],
        [7.7000, 3.0000, 6.1000, 2.3000],
        [6.3000, 3.4000, 5.6000, 2.4000],
        [6.4000, 3.1000, 5.5000, 1.8000],
        [6.0000, 3.0000, 4.8000, 1.8000],
        [6.9000, 3.1000, 5.4000, 2.1000],
        [6.7000, 3.1000, 5.6000, 2.4000],
        [6.9000, 3.1000, 5.1000, 2.3000],
        [5.8000, 2.7000, 5.1000, 1.9000],
        [6.8000, 3.2000, 5.9000, 2.3000],
        [6.7000, 3.3000, 5.7000, 2.5000],
        [6.7000, 3.0000, 5.2000, 2.3000],
        [6.3000, 2.5000, 5.0000, 1.9000],
        [6.5000, 3.0000, 5.2000, 2.0000],
        [6.2000, 3.4000, 5.4000, 2.3000],
        [5.9000, 3.0000, 5.1000, 1.8000]], dtype=torch.float64)
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2])
Parameter containing:
tensor([[-0.0560,  0.4659,  0.1224,  0.4361]], requires_grad=True)
Parameter containing:
tensor([0.0210], requires_grad=True)
Epoch [10/100], Loss: 1.0411
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:63: UserWarning: Using a target size
(torch.Size([5])) that is different to the input size (torch.Size([5, 1])). This will likely lead to incorrect results
due to broadcasting. Please ensure they have the same size.
Epoch [20/100], Loss: 0.8621
Epoch [30/100], Loss: 0.7811
Epoch [40/100], Loss: 0.4791
Epoch [50/100], Loss: 1.2322
Epoch [60/100], Loss: 0.8519
Epoch [70/100], Loss: 0.8905
```

Epoch [80/100], Loss: 0.8590
Epoch [90/100], Loss: 0.6565
Epoch [100/100], Loss: 0.8635
tensor([[0.7880],
[0.6137],
[0.7230],
[0.7075],
[0.8451],
[1.0022],
[0.8673],
[0.7676],
[0.6519],
[0.6188],
[0.8277],
[0.8044],
[0.5885],
[0.6531],
[0.8730],
[1.1507],
[0.9824],
[0.8291],
[0.8723],
[0.9577],
[0.7140],
[0.9575],
[0.8889],
[0.8436],
[0.8193],
[0.6077],
[0.8547],
[0.7771],
[0.7309],
[0.7378],
[0.6807],
[0.7862],
[0.9834],
[1.0130],
[0.6599],
[0.6703],
[0.7195],
[0.8200],
[0.6882],
[0.7517],
[0.8400],
[0.4248],
[0.7706],
[0.9781],
[1.0186],
[0.6706],
[0.9216],
[0.7438],
[0.8436],
[0.7215],
[1.0186],
[1.1451],

[1.0443],
[0.8103],
[0.9693],
[1.0094],
[1.2532],
[0.7890],
[0.9125],
[1.0590],
[0.6181],
[1.1273],
[0.5664],
[1.0380],
[1.0219],
[1.0102],
[1.1899],
[0.8093],
[0.7647],
[0.7898],
[1.3627],
[0.9210],
[0.8923],
[0.9147],
[0.9294],
[0.9849],
[0.8905],
[1.1219],
[1.0851],
[0.7542],
[0.7595],
[0.7134],
[0.8815],
[1.0734],
[1.2216],
[1.3322],
[1.0661],
[0.7029],
[1.0879],
[0.8927],
[0.9127],
[1.0743],
[0.8452],
[0.7319],
[0.9692],
[1.0359],
[1.0357],
[0.9612],
[0.8246],
[0.9896],
[1.6872],
[1.2284],
[1.2672],
[1.2151],
[1.3987],
[1.2224],
[1.1771],

[1.0908],
[0.9965],
[1.6728],
[1.3643],
[1.1429],
[1.2950],
[1.1979],
[1.4749],
[1.5133],
[1.2195],
[1.5823],
[1.1386],
[0.8213],
[1.4536],
[1.3325],
[1.0880],
[1.0979],
[1.4445],
[1.2155],
[1.1501],
[1.2534],
[1.2811],
[1.0410],
[1.0648],
[1.4535],
[1.3222],
[1.0259],
[0.9589],
[1.2639],
[1.6676],
[1.2767],
[1.2643],
[1.3154],
[1.4803],
[1.3827],
[1.2284],
[1.4794],
[1.6088],
[1.3782],
[1.0615],
[1.2868],
[1.6325],
[1.2951]], grad_fn=<AddmmBackward0>)