In [1]:

```
import torch
import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt

df = pd.DataFrame(pd.read_csv("Housing.csv"))
df.replace(('yes', 'no'), (1, 0), inplace=True)
df.replace(('furnished', 'semi-furnished', 'unfurnished'), (1, .5, 0), inplace=True)
housing_N=(df-df.min())/(df.max()-df.min())
housing_N = housing_N[['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']]
#print(housing_N.head())

housing_N = housing_N.astype('float32')

t_un = housing_N.drop('price', axis=1, )
print(type(t_un))
t_c = housing_N['price']
t_un = torch.from_numpy(t_un.to_numpy())
t_c = torch.from_numpy(t_c.to_numpy())
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [2]:

```
import torch
print(t_un.shape)
n_samples = t_un.shape[0]
n_val = int(.2 * n_samples)
shuffled_indices = torch.randperm(n_samples).numpy()
train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]
print(type(train_indices))
```

```
torch.Size([545, 5])
<class 'numpy.ndarray'>
```

In [3]:

```
train_t_un = t_un[train_indices]
train_t_c = t_c[train_indices]

val_t_un = t_un[val_indices]
val_t_c = t_c[val_indices]
print(train_t_un.shape)
```

```
torch.Size([436, 5])
```

In [4]:

```
import torch.nn as nn
seq_model = nn.Sequential(
            nn.Linear(5, 8),
            nn.Tanh(),
            nn.Linear(8, 1))
seq_model
```

Out[4]:

```
Sequential(
  (0): Linear(in_features=5, out_features=8, bias=True)
  (1): Tanh()
  (2): Linear(in_features=8, out_features=1, bias=True)
)
```

In [5]:

```
import torch.optim as optim
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
optimizer = optim.SGD(seq_model.parameters(), lr=1e-2)
```

```python
def training_loop(n_epochs, optimizer, model, loss_fn, t_un_train, t_un_val, t_c_train, t_
    t_c_val = torch.unsqueeze(t_c_val, dim=-1)
    t_c_train = torch.unsqueeze(t_c_train, dim=-1)
    for epoch in range(1, n_epochs+1):
        t_p_train = model(t_un_train)
        #print(t_c_train.shape)
        loss_train = loss_fn(t_p_train, t_c_train)
        t_p_val = model(t_un_val)


        loss_val = loss_fn(t_p_val, t_c_val)

        optimizer.zero_grad()
        loss_train.backward()
        optimizer.step()
        if epoch <  5 or epoch == 200:
            print(f"Epoch {epoch}, Training Loss {loss_train.item():.4f},"f" Validation lo
```

In [6]:

```python
import time
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
seq_model.to(device)
train_t_un = train_t_un.to(device)
val_t_un = val_t_un.to(device)
train_t_c = train_t_c.to(device)
val_t_c = val_t_c.to(device)
print(train_t_un.is_cuda)
start = time.time()
training_loop(n_epochs= 200, optimizer = optimizer, model = seq_model, loss_fn = nn.MSELos
t_un_train=train_t_un, t_un_val = val_t_un, t_c_train = train_t_c, t_c_val = val_t_c)
end = time.time()
print(end - start)
```

```
True
Epoch 1, Training Loss 0.1144, Validation loss 0.1138
Epoch 2, Training Loss 0.1076, Validation loss 0.1072
Epoch 3, Training Loss 0.1014, Validation loss 0.1010
Epoch 4, Training Loss 0.0957, Validation loss 0.0954
Epoch 200, Training Loss 0.0232, Validation loss 0.0245
1.5103340148925781
```

In [7]:

```python
from ptflops import get_model_complexity_info


macs, params = get_model_complexity_info(seq_model, (436, 5), as_strings=True, print_per_l
print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
print('{:<30}  {:<8}'.format('Number of parameters: ', params))
```

```
Computational complexity:       0.0 GMac
Number of parameters:           57
```

In [8]:

```python
with torch.no_grad():
   #seq_model.eval()
    avg = 0
    pred = []
    for val in val_t_un:
        pred.append(seq_model(val))
    i = 0
    for tar in val_t_c:
        temp = pred[i]/tar
        if temp > 1:
```

```
            temp = 1 - (temp - 1)
        if temp > 2:
            temp = 1 - (temp - 2)
        avg += abs(temp)
        i += 1
    ttl_avg = avg / len(val_t_un)
print(ttl_avg[0].item())
```

```
0.8252021074295044
```

In [9]:

```
#PART B Model
import time
seq_model2 = nn.Sequential(
            nn.Linear(5, 256),
            nn.Tanh(),
            nn.Linear(256, 128),
            nn.Tanh(),
            nn.Linear(128, 64),
            nn.Tanh(),
            nn.Linear(64, 1))
seq_model2.to(device)
start = time.time()
optimizer3 = optim.SGD(seq_model2.parameters(), lr=1e-2)
training_loop(n_epochs= 200, optimizer = optimizer3, model = seq_model2, loss_fn = nn.MSEI
t_un_train=train_t_un, t_un_val = val_t_un, t_c_train = train_t_c, t_c_val = val_t_c)
end = time.time()
print(end - start)
```

```
Epoch 1, Training Loss 0.0565, Validation loss 0.0578
Epoch 2, Training Loss 0.0500, Validation loss 0.0514
Epoch 3, Training Loss 0.0448, Validation loss 0.0463
Epoch 4, Training Loss 0.0406, Validation loss 0.0422
Epoch 200, Training Loss 0.0157, Validation loss 0.0167
0.33489227294921875
```

In [10]:

```
from ptflops import get_model_complexity_info


macs, params = get_model_complexity_info(seq_model2, (465, 5), as_strings=True, print_per_
print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
print('{:<30}  {:<8}'.format('Number of parameters: ', params))
```

```
Computational complexity:       0.02 GMac
Number of parameters:           42.75 k
```

In [11]:

```
with torch.no_grad():
   #seq_model.eval()
    avg = 0
    pred = []
    for val in val_t_un:
        pred.append(seq_model2(val))
    i = 0
    for tar in val_t_c:
        temp = pred[i]/tar
        if temp > 1:
            temp = 1 - (temp - 1)
        if temp > 2:
            temp = 1 - (temp - 2)
        avg += abs(temp)
        i += 1
    ttl_avg = avg / len(val_t_un)
print(abs(ttl_avg[0].item()))
```

```
0.8387661576271057
```

In [12]:

```
# PART 2
import ssl
import torch
ssl._create_default_https_context = ssl._create_unverified_context
from torchvision import datasets
data_path = "./data/"
cifar10 = datasets.CIFAR10(data_path, train=True, download=True)
cifar10_val = datasets.CIFAR10(data_path, train=False, download=True)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [13]:

```
from torchvision import transforms
tensor_cifar10 = datasets.CIFAR10(data_path, train=True, download=False, transform=transfo
imgs = torch.stack([img_t for img_t, _ in tensor_cifar10], dim=3)
imgs.shape
tensor_cifar10_val = datasets.CIFAR10(data_path, train=False, download=False, transform=tr
imgs_val = torch.stack([img_t for img_t, _ in tensor_cifar10_val], dim=3)
imgs_val.shape
```

Out[13]:

```
torch.Size([3, 32, 32, 10000])
```

In [14]:

```
imgs_val.view(3, -1).mean(dim=1)
```

Out[14]:

```
tensor([0.4942, 0.4851, 0.4504])
```

In [15]:

```
imgs_val.view(3, -1).std(dim=1)
```

Out[15]:

```
tensor([0.2467, 0.2429, 0.2616])
```

In [16]:

```
imgs.view(3, -1).mean(dim=1)
```

Out[16]:

```
tensor([0.4914, 0.4822, 0.4465])
```

In [17]:

```
imgs.view(3, -1).std(dim=1)
```

Out[17]:

```
tensor([0.2470, 0.2435, 0.2616])
```

In [18]:

```
transforms.Normalize((0.4914, 0.4822, 0.4465),  (0.2470, 0.2435, 0.2616))
```

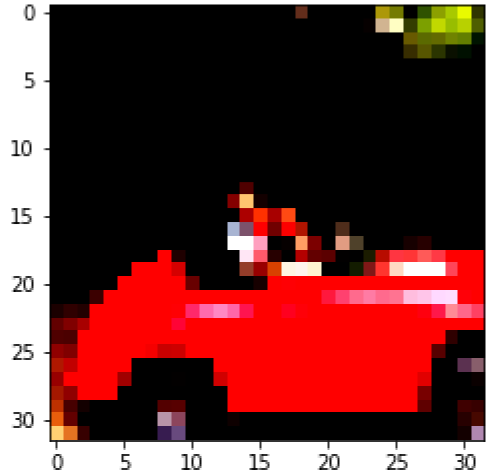
Out[18]:

```
Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.247, 0.2435, 0.2616))
```

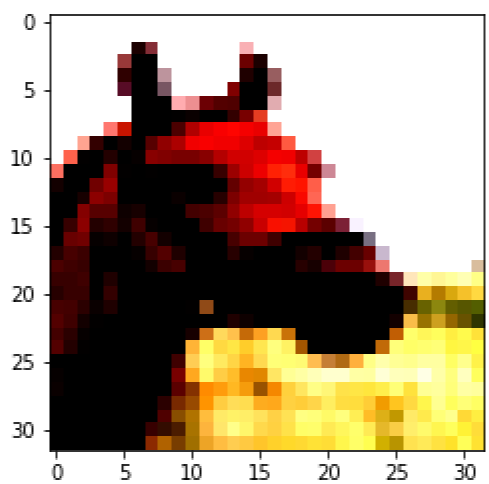
In [19]:

```
import matplotlib.pyplot as plt
transformed_cifar10 = datasets.CIFAR10(data_path, train=True, download=False, transform=tr
img_t, _  = transformed_cifar10[99]
plt.imshow(img_t.permute(1, 2, 0))
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
```



In [20]:
```
transforms.Normalize((0.4942, 0.4851, 0.4504),  (0.2467, 0.2429, 0.2616))
```
Out[20]:
```
Normalize(mean=(0.4942, 0.4851, 0.4504), std=(0.2467, 0.2429, 0.2616))
```

In [21]:
```
val_transformed_cifar10 = datasets.CIFAR10(data_path, train=False, download=False, transfo
img_t, _  = val_transformed_cifar10[99]
plt.imshow(img_t.permute(1, 2, 0))
plt.show()
```
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
```


In [22]:
```
import torch.nn as nn
import torch.optim as optim
print(torch.cuda.is_available())
import time
device = torch.device('cuda:0')
kwargs = {'num_workers': 1, 'pin_memory': True}
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64, shuffle=Tru
model = nn.Sequential(nn.Linear(3072, 512), nn.Tanh(), nn.Linear(512, 10), nn.LogSoftmax(d
model.to(device)
#transformed_cifar10.to(device)
loss_fn = nn.NLLLoss()
learning_rate = 1e-2
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
n_epochs = 300

start = time.time()

for epoch in range(n_epochs):
```

```
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)
        batch_size = imgs.shape[0]
        outputs = model(imgs.view(batch_size, -1).to(device))
        loss = loss_fn(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("Epoch: %d, Loss: %f" % (epoch, float(loss)))
end = time.time()
print(end - start)
```

```
True
Epoch: 0, Loss: 1.823992
Epoch: 1, Loss: 1.665446
Epoch: 2, Loss: 1.340552
Epoch: 3, Loss: 1.747149
Epoch: 4, Loss: 1.285698
Epoch: 5, Loss: 1.828456
Epoch: 6, Loss: 2.020544
Epoch: 7, Loss: 1.096881
Epoch: 8, Loss: 1.082237
Epoch: 9, Loss: 1.975957
Epoch: 10, Loss: 1.205648
Epoch: 11, Loss: 1.182079
Epoch: 12, Loss: 1.369270
Epoch: 13, Loss: 1.483496
Epoch: 14, Loss: 1.058547
Epoch: 15, Loss: 1.223945
Epoch: 16, Loss: 0.930954
Epoch: 17, Loss: 1.103109
Epoch: 18, Loss: 1.260656
Epoch: 19, Loss: 0.951822
Epoch: 20, Loss: 0.960585
Epoch: 21, Loss: 0.947495
Epoch: 22, Loss: 1.764741
Epoch: 23, Loss: 0.910077
Epoch: 24, Loss: 0.807602
Epoch: 25, Loss: 0.605959
Epoch: 26, Loss: 0.964515
Epoch: 27, Loss: 1.080329
Epoch: 28, Loss: 0.963439
Epoch: 29, Loss: 0.590387
Epoch: 30, Loss: 0.734520
Epoch: 31, Loss: 0.693765
Epoch: 32, Loss: 0.558971
Epoch: 33, Loss: 0.673594
Epoch: 34, Loss: 0.593907
Epoch: 35, Loss: 0.305203
Epoch: 36, Loss: 0.410466
Epoch: 37, Loss: 0.512907
Epoch: 38, Loss: 0.321848
Epoch: 39, Loss: 0.489095
Epoch: 40, Loss: 0.778805
Epoch: 41, Loss: 0.677711
Epoch: 42, Loss: 0.353084
Epoch: 43, Loss: 0.176177
Epoch: 44, Loss: 0.291422
Epoch: 45, Loss: 0.334004
Epoch: 46, Loss: 0.273592
Epoch: 47, Loss: 0.532092
Epoch: 48, Loss: 0.468087
Epoch: 49, Loss: 0.170093
```

```
Epoch: 50, Loss: 0.373186
Epoch: 51, Loss: 0.309941
Epoch: 52, Loss: 0.433412
Epoch: 53, Loss: 0.235918
Epoch: 54, Loss: 0.195996
Epoch: 55, Loss: 0.276058
Epoch: 56, Loss: 0.406309
Epoch: 57, Loss: 0.142081
Epoch: 58, Loss: 0.218915
Epoch: 59, Loss: 0.189670
Epoch: 60, Loss: 0.183924
Epoch: 61, Loss: 0.273308
Epoch: 62, Loss: 0.123961
Epoch: 63, Loss: 0.105437
Epoch: 64, Loss: 0.248037
Epoch: 65, Loss: 0.189067
Epoch: 66, Loss: 0.135403
Epoch: 67, Loss: 0.084875
Epoch: 68, Loss: 0.101953
Epoch: 69, Loss: 0.247039
Epoch: 70, Loss: 0.187243
Epoch: 71, Loss: 0.070558
Epoch: 72, Loss: 0.121030
Epoch: 73, Loss: 0.054968
Epoch: 74, Loss: 0.102090
Epoch: 75, Loss: 0.063454
Epoch: 76, Loss: 0.066198
Epoch: 77, Loss: 0.111365
Epoch: 78, Loss: 0.084349
Epoch: 79, Loss: 0.075002
Epoch: 80, Loss: 0.100341
Epoch: 81, Loss: 0.071044
Epoch: 82, Loss: 0.114674
Epoch: 83, Loss: 0.046078
Epoch: 84, Loss: 0.062157
Epoch: 85, Loss: 0.105139
Epoch: 86, Loss: 0.074049
Epoch: 87, Loss: 0.069959
Epoch: 88, Loss: 0.050594
Epoch: 89, Loss: 0.068399
Epoch: 90, Loss: 0.055091
Epoch: 91, Loss: 0.038051
Epoch: 92, Loss: 0.060497
Epoch: 93, Loss: 0.063907
Epoch: 94, Loss: 0.058012
Epoch: 95, Loss: 0.079084
Epoch: 96, Loss: 0.043046
Epoch: 97, Loss: 0.049806
Epoch: 98, Loss: 0.038109
Epoch: 99, Loss: 0.042486
Epoch: 100, Loss: 0.034576
Epoch: 101, Loss: 0.043884
Epoch: 102, Loss: 0.116956
Epoch: 103, Loss: 0.086062
Epoch: 104, Loss: 0.035196
Epoch: 105, Loss: 0.067843
Epoch: 106, Loss: 0.026262
Epoch: 107, Loss: 0.138624
Epoch: 108, Loss: 0.038550
Epoch: 109, Loss: 0.028477
Epoch: 110, Loss: 0.020315
Epoch: 111, Loss: 0.033050
Epoch: 112, Loss: 0.024929
Epoch: 113, Loss: 0.021515
Epoch: 114, Loss: 0.039477
Epoch: 115, Loss: 0.029767
```

```
Epoch: 116, Loss: 0.033796
Epoch: 117, Loss: 0.033473
Epoch: 118, Loss: 0.038767
Epoch: 119, Loss: 0.032873
Epoch: 120, Loss: 0.035481
Epoch: 121, Loss: 0.029208
Epoch: 122, Loss: 0.022991
Epoch: 123, Loss: 0.025108
Epoch: 124, Loss: 0.034725
Epoch: 125, Loss: 0.077776
Epoch: 126, Loss: 0.029488
Epoch: 127, Loss: 0.020204
Epoch: 128, Loss: 0.013263
Epoch: 129, Loss: 0.023135
Epoch: 130, Loss: 0.023173
Epoch: 131, Loss: 0.026625
Epoch: 132, Loss: 0.024176
Epoch: 133, Loss: 0.032794
Epoch: 134, Loss: 0.015031
Epoch: 135, Loss: 0.013592
Epoch: 136, Loss: 0.024928
Epoch: 137, Loss: 0.017352
Epoch: 138, Loss: 0.018429
Epoch: 139, Loss: 0.017609
Epoch: 140, Loss: 0.025735
Epoch: 141, Loss: 0.041591
Epoch: 142, Loss: 0.029601
Epoch: 143, Loss: 0.016587
Epoch: 144, Loss: 0.013326
Epoch: 145, Loss: 0.028634
Epoch: 146, Loss: 0.023978
Epoch: 147, Loss: 0.052346
Epoch: 148, Loss: 0.029793
Epoch: 149, Loss: 0.013242
Epoch: 150, Loss: 0.015031
Epoch: 151, Loss: 0.017848
Epoch: 152, Loss: 0.009876
Epoch: 153, Loss: 0.019166
Epoch: 154, Loss: 0.022139
Epoch: 155, Loss: 0.016611
Epoch: 156, Loss: 0.019621
Epoch: 157, Loss: 0.015717
Epoch: 158, Loss: 0.021759
Epoch: 159, Loss: 0.011691
Epoch: 160, Loss: 0.035024
Epoch: 161, Loss: 0.018283
Epoch: 162, Loss: 0.046963
Epoch: 163, Loss: 0.016430
Epoch: 164, Loss: 0.023440
Epoch: 165, Loss: 0.014861
Epoch: 166, Loss: 0.020913
Epoch: 167, Loss: 0.016771
Epoch: 168, Loss: 0.013153
Epoch: 169, Loss: 0.005344
Epoch: 170, Loss: 0.016742
Epoch: 171, Loss: 0.013914
Epoch: 172, Loss: 0.007528
Epoch: 173, Loss: 0.013282
Epoch: 174, Loss: 0.013953
Epoch: 175, Loss: 0.013372
Epoch: 176, Loss: 0.011807
Epoch: 177, Loss: 0.016644
Epoch: 178, Loss: 0.011619
Epoch: 179, Loss: 0.016104
Epoch: 180, Loss: 0.011398
Epoch: 181, Loss: 0.019483
```

```
Epoch: 182, Loss: 0.006982
Epoch: 183, Loss: 0.015431
Epoch: 184, Loss: 0.015878
Epoch: 185, Loss: 0.013732
Epoch: 186, Loss: 0.017885
Epoch: 187, Loss: 0.011979
Epoch: 188, Loss: 0.008570
Epoch: 189, Loss: 0.007895
Epoch: 190, Loss: 0.008546
Epoch: 191, Loss: 0.010546
Epoch: 192, Loss: 0.010734
Epoch: 193, Loss: 0.048393
Epoch: 194, Loss: 0.008738
Epoch: 195, Loss: 0.011596
Epoch: 196, Loss: 0.009833
Epoch: 197, Loss: 0.006388
Epoch: 198, Loss: 0.011525
Epoch: 199, Loss: 0.012214
Epoch: 200, Loss: 0.007747
Epoch: 201, Loss: 0.010553
Epoch: 202, Loss: 0.008837
Epoch: 203, Loss: 0.010749
Epoch: 204, Loss: 0.010851
Epoch: 205, Loss: 0.012867
Epoch: 206, Loss: 0.008875
Epoch: 207, Loss: 0.006454
Epoch: 208, Loss: 0.014278
Epoch: 209, Loss: 0.012239
Epoch: 210, Loss: 0.009041
Epoch: 211, Loss: 0.007543
Epoch: 212, Loss: 0.006148
Epoch: 213, Loss: 0.008415
Epoch: 214, Loss: 0.008467
Epoch: 215, Loss: 0.011042
Epoch: 216, Loss: 0.010561
Epoch: 217, Loss: 0.010678
Epoch: 218, Loss: 0.012627
Epoch: 219, Loss: 0.006573
Epoch: 220, Loss: 0.007724
Epoch: 221, Loss: 0.007604
Epoch: 222, Loss: 0.008883
Epoch: 223, Loss: 0.014918
Epoch: 224, Loss: 0.020283
Epoch: 225, Loss: 0.009974
Epoch: 226, Loss: 0.012318
Epoch: 227, Loss: 0.005976
Epoch: 228, Loss: 0.008650
Epoch: 229, Loss: 0.010217
Epoch: 230, Loss: 0.007773
Epoch: 231, Loss: 0.006352
Epoch: 232, Loss: 0.006796
Epoch: 233, Loss: 0.008625
Epoch: 234, Loss: 0.009623
Epoch: 235, Loss: 0.009656
Epoch: 236, Loss: 0.008805
Epoch: 237, Loss: 0.010900
Epoch: 238, Loss: 0.012859
Epoch: 239, Loss: 0.006009
Epoch: 240, Loss: 0.006394
Epoch: 241, Loss: 0.005619
Epoch: 242, Loss: 0.009582
Epoch: 243, Loss: 0.008709
Epoch: 244, Loss: 0.009095
Epoch: 245, Loss: 0.008909
Epoch: 246, Loss: 0.008938
Epoch: 247, Loss: 0.009630
```

```
Epoch: 248, Loss: 0.007468
Epoch: 249, Loss: 0.006255
Epoch: 250, Loss: 0.017256
Epoch: 251, Loss: 0.011347
Epoch: 252, Loss: 0.007097
Epoch: 253, Loss: 0.008163
Epoch: 254, Loss: 0.004110
Epoch: 255, Loss: 0.004506
Epoch: 256, Loss: 0.005345
Epoch: 257, Loss: 0.008415
Epoch: 258, Loss: 0.005728
Epoch: 259, Loss: 0.007193
Epoch: 260, Loss: 0.008263
Epoch: 261, Loss: 0.007632
Epoch: 262, Loss: 0.012011
Epoch: 263, Loss: 0.006647
Epoch: 264, Loss: 0.006725
Epoch: 265, Loss: 0.005302
Epoch: 266, Loss: 0.006868
Epoch: 267, Loss: 0.007628
Epoch: 268, Loss: 0.008539
Epoch: 269, Loss: 0.007235
Epoch: 270, Loss: 0.006279
Epoch: 271, Loss: 0.005989
Epoch: 272, Loss: 0.007736
Epoch: 273, Loss: 0.008202
Epoch: 274, Loss: 0.006576
Epoch: 275, Loss: 0.004839
Epoch: 276, Loss: 0.005246
Epoch: 277, Loss: 0.004874
Epoch: 278, Loss: 0.006574
Epoch: 279, Loss: 0.006809
Epoch: 280, Loss: 0.004787
Epoch: 281, Loss: 0.005013
Epoch: 282, Loss: 0.009218
Epoch: 283, Loss: 0.006538
Epoch: 284, Loss: 0.008602
Epoch: 285, Loss: 0.009340
Epoch: 286, Loss: 0.006806
Epoch: 287, Loss: 0.004566
Epoch: 288, Loss: 0.005170
Epoch: 289, Loss: 0.007265
Epoch: 290, Loss: 0.007128
Epoch: 291, Loss: 0.005895
Epoch: 292, Loss: 0.007114
Epoch: 293, Loss: 0.009223
Epoch: 294, Loss: 0.006780
Epoch: 295, Loss: 0.006364
Epoch: 296, Loss: 0.006582
Epoch: 297, Loss: 0.008454
Epoch: 298, Loss: 0.006042
Epoch: 299, Loss: 0.005357
2869.39600276947
```

In [23]:

```
from ptflops import get_model_complexity_info


macs, params = get_model_complexity_info(model, (1, 3072), as_strings=True, print_per_laye
print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
print('{:<30}  {:<8}'.format('Number of parameters: ', params))
```

```
Computational complexity:       0.0 GMac
Number of parameters:           1.58 M
```

In [24]:

```
val_loader = torch.utils.data.DataLoader(val_transformed_cifar10, batch_size=64, shuffle=F

correct = 0
total = 0
with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)
        batch_size=imgs.shape[0]
        outputs = model(imgs.view(batch_size, -1))
        _, predicted = torch.max(outputs, dim=1)
        total += labels.shape[0]
        correct += int((predicted==labels).sum())
    print("Accuracy %f", correct/total)
```

```
Accuracy %f 0.4624
```

In [25]:

```
import torch.nn as nn
import torch.optim as optim
print(torch.cuda.is_available())
device = torch.device('cuda:0')
kwargs = {'num_workers': 1, 'pin_memory': True}
train_loader = torch.utils.data.DataLoader(transformed_cifar10, batch_size=64, shuffle=Tru
model2 = nn.Sequential(nn.Linear(3072, 512), nn.Tanh(), nn.Linear(512, 256), nn.Tanh(), nn
model2.to(device)
#transformed_cifar10.to(device)
loss_fn = nn.NLLLoss()
learning_rate = 1e-2
optimizer = optim.SGD(model2.parameters(), lr=learning_rate)
#optimizer.zero_grad()
n_epochs = 200
start = time.time()
for epoch in range(n_epochs):
    for imgs, labels in train_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)
        batch_size = imgs.shape[0]
        outputs = model2(imgs.view(batch_size, -1).to(device))
        loss = loss_fn(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("Epoch: %d, Loss: %f" % (epoch, float(loss)))
end = time.time()
print(end - start)
```

```
True
Epoch: 0, Loss: 1.707413
Epoch: 1, Loss: 1.966603
Epoch: 2, Loss: 1.907584
Epoch: 3, Loss: 1.549764
Epoch: 4, Loss: 1.524193
Epoch: 5, Loss: 1.288348
Epoch: 6, Loss: 1.849928
Epoch: 7, Loss: 1.509243
Epoch: 8, Loss: 1.489519
Epoch: 9, Loss: 1.793851
Epoch: 10, Loss: 0.734490
Epoch: 11, Loss: 1.398465
Epoch: 12, Loss: 1.568537
Epoch: 13, Loss: 1.027514
Epoch: 14, Loss: 1.203753
Epoch: 15, Loss: 1.250190
Epoch: 16, Loss: 0.779541
```

```
Epoch: 17, Loss: 1.304618
Epoch: 18, Loss: 1.302351
Epoch: 19, Loss: 1.175987
Epoch: 20, Loss: 0.632308
Epoch: 21, Loss: 0.946094
Epoch: 22, Loss: 0.542526
Epoch: 23, Loss: 1.823120
Epoch: 24, Loss: 0.829579
Epoch: 25, Loss: 0.943334
Epoch: 26, Loss: 0.444218
Epoch: 27, Loss: 0.459384
Epoch: 28, Loss: 0.426672
Epoch: 29, Loss: 0.605277
Epoch: 30, Loss: 0.902058
Epoch: 31, Loss: 0.559479
Epoch: 32, Loss: 0.606705
Epoch: 33, Loss: 0.544501
Epoch: 34, Loss: 0.356462
Epoch: 35, Loss: 0.355857
Epoch: 36, Loss: 0.409773
Epoch: 37, Loss: 0.188113
Epoch: 38, Loss: 0.200532
Epoch: 39, Loss: 0.171625
Epoch: 40, Loss: 0.167427
Epoch: 41, Loss: 0.219591
Epoch: 42, Loss: 0.059690
Epoch: 43, Loss: 0.063790
Epoch: 44, Loss: 0.248724
Epoch: 45, Loss: 0.058901
Epoch: 46, Loss: 0.123478
Epoch: 47, Loss: 0.047628
Epoch: 48, Loss: 0.425450
Epoch: 49, Loss: 0.135685
Epoch: 50, Loss: 0.059067
Epoch: 51, Loss: 0.098424
Epoch: 52, Loss: 0.098628
Epoch: 53, Loss: 0.167175
Epoch: 54, Loss: 0.056608
Epoch: 55, Loss: 0.176471
Epoch: 56, Loss: 0.129674
Epoch: 57, Loss: 0.026208
Epoch: 58, Loss: 0.025535
Epoch: 59, Loss: 0.021906
Epoch: 60, Loss: 0.017154
Epoch: 61, Loss: 0.020460
Epoch: 62, Loss: 0.038585
Epoch: 63, Loss: 0.007547
Epoch: 64, Loss: 0.007694
Epoch: 65, Loss: 0.022384
Epoch: 66, Loss: 0.005091
Epoch: 67, Loss: 0.014794
Epoch: 68, Loss: 0.320011
Epoch: 69, Loss: 0.228721
Epoch: 70, Loss: 0.005759
Epoch: 71, Loss: 0.007874
Epoch: 72, Loss: 0.015649
Epoch: 73, Loss: 0.005663
Epoch: 74, Loss: 0.003334
Epoch: 75, Loss: 0.005894
Epoch: 76, Loss: 0.004422
Epoch: 77, Loss: 0.002575
Epoch: 78, Loss: 0.002054
Epoch: 79, Loss: 0.002374
Epoch: 80, Loss: 0.004921
Epoch: 81, Loss: 0.006709
Epoch: 82, Loss: 0.003971
```

```
Epoch: 83, Loss: 0.003380
Epoch: 84, Loss: 0.002170
Epoch: 85, Loss: 0.003490
Epoch: 86, Loss: 0.002095
Epoch: 87, Loss: 0.004882
Epoch: 88, Loss: 0.003704
Epoch: 89, Loss: 0.002829
Epoch: 90, Loss: 0.003503
Epoch: 91, Loss: 0.003522
Epoch: 92, Loss: 0.001138
Epoch: 93, Loss: 0.002988
Epoch: 94, Loss: 0.002277
Epoch: 95, Loss: 0.001075
Epoch: 96, Loss: 0.002372
Epoch: 97, Loss: 0.000925
Epoch: 98, Loss: 0.000887
Epoch: 99, Loss: 0.002154
Epoch: 100, Loss: 0.001355
Epoch: 101, Loss: 0.002081
Epoch: 102, Loss: 0.000939
Epoch: 103, Loss: 0.002174
Epoch: 104, Loss: 0.001916
Epoch: 105, Loss: 0.001542
Epoch: 106, Loss: 0.001771
Epoch: 107, Loss: 0.001269
Epoch: 108, Loss: 0.003195
Epoch: 109, Loss: 0.000863
Epoch: 110, Loss: 0.001231
Epoch: 111, Loss: 0.001770
Epoch: 112, Loss: 0.001281
Epoch: 113, Loss: 0.001214
Epoch: 114, Loss: 0.002428
Epoch: 115, Loss: 0.001718
Epoch: 116, Loss: 0.000854
Epoch: 117, Loss: 0.001705
Epoch: 118, Loss: 0.001076
Epoch: 119, Loss: 0.001184
Epoch: 120, Loss: 0.000864
Epoch: 121, Loss: 0.000869
Epoch: 122, Loss: 0.000752
Epoch: 123, Loss: 0.002273
Epoch: 124, Loss: 0.001008
Epoch: 125, Loss: 0.001249
Epoch: 126, Loss: 0.001193
Epoch: 127, Loss: 0.002695
Epoch: 128, Loss: 0.000818
Epoch: 129, Loss: 0.001629
Epoch: 130, Loss: 0.000947
Epoch: 131, Loss: 0.000968
Epoch: 132, Loss: 0.001147
Epoch: 133, Loss: 0.000880
Epoch: 134, Loss: 0.000875
Epoch: 135, Loss: 0.001060
Epoch: 136, Loss: 0.001059
Epoch: 137, Loss: 0.001306
Epoch: 138, Loss: 0.001110
Epoch: 139, Loss: 0.000469
Epoch: 140, Loss: 0.001196
Epoch: 141, Loss: 0.000687
Epoch: 142, Loss: 0.000983
Epoch: 143, Loss: 0.000440
Epoch: 144, Loss: 0.001039
Epoch: 145, Loss: 0.001334
Epoch: 146, Loss: 0.000443
Epoch: 147, Loss: 0.000310
Epoch: 148, Loss: 0.000591
```

```
Epoch: 149, Loss: 0.001082
Epoch: 150, Loss: 0.000690
Epoch: 151, Loss: 0.000831
Epoch: 152, Loss: 0.001828
Epoch: 153, Loss: 0.000994
Epoch: 154, Loss: 0.000739
Epoch: 155, Loss: 0.000727
Epoch: 156, Loss: 0.000739
Epoch: 157, Loss: 0.001105
Epoch: 158, Loss: 0.000926
Epoch: 159, Loss: 0.001294
Epoch: 160, Loss: 0.001022
Epoch: 161, Loss: 0.000891
Epoch: 162, Loss: 0.000470
Epoch: 163, Loss: 0.000985
Epoch: 164, Loss: 0.000688
Epoch: 165, Loss: 0.000935
Epoch: 166, Loss: 0.000700
Epoch: 167, Loss: 0.000974
Epoch: 168, Loss: 0.000549
Epoch: 169, Loss: 0.000756
Epoch: 170, Loss: 0.001196
Epoch: 171, Loss: 0.000551
Epoch: 172, Loss: 0.001030
Epoch: 173, Loss: 0.000581
Epoch: 174, Loss: 0.000651
Epoch: 175, Loss: 0.000973
Epoch: 176, Loss: 0.000465
Epoch: 177, Loss: 0.000301
Epoch: 178, Loss: 0.000355
Epoch: 179, Loss: 0.001089
Epoch: 180, Loss: 0.000290
Epoch: 181, Loss: 0.000982
Epoch: 182, Loss: 0.000877
Epoch: 183, Loss: 0.000406
Epoch: 184, Loss: 0.000923
Epoch: 185, Loss: 0.000898
Epoch: 186, Loss: 0.000770
Epoch: 187, Loss: 0.000545
Epoch: 188, Loss: 0.000816
Epoch: 189, Loss: 0.000803
Epoch: 190, Loss: 0.000315
Epoch: 191, Loss: 0.000632
Epoch: 192, Loss: 0.000538
Epoch: 193, Loss: 0.000601
Epoch: 194, Loss: 0.000517
Epoch: 195, Loss: 0.000606
Epoch: 196, Loss: 0.000649
Epoch: 197, Loss: 0.000385
Epoch: 198, Loss: 0.000924
Epoch: 199, Loss: 0.000568
1912.4364964962006
```

In [26]:

```
macs, params = get_model_complexity_info(model, (1, 3072),  as_strings=True, print_per_lay
print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
print('{:<30}  {:<8}'.format('Number of parameters: ', params))
```

```
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Warning: variables __flops__ or __params__ are already defined for the moduleLinear ptflop
s can affect your code!
Computational complexity:       0.0 GMac
Number of parameters:           1.58 M
```

In [27]:

```python
#torch.save(model2.state_dict(), 'model2.pt')
```

In [28]:

```python
val_loader = torch.utils.data.DataLoader(val_transformed_cifar10, batch_size=64, shuffle=F

correct = 0
total = 0
with torch.no_grad():
    for imgs, labels in val_loader:
        imgs = imgs.to(device)
        labels = labels.to(device)
        batch_size=imgs.shape[0]
        outputs = model2(imgs.view(batch_size, -1))
        _, predicted = torch.max(outputs, dim=1)
        total += labels.shape[0]
        correct += int((predicted==labels).sum())
    print("Accuracy %f", correct/total)
```

```
Accuracy %f 0.4596
```

In [ ]:

In [ ]: