Problem 1 A

Friday, January 27, 2023 10:41 AM

$$f_W(X) = ABX$$

$1\times n$, $1\times k$, $k\times d$, $d\times n$

$$= [A_1 A_2 \cdots A_k] \begin{bmatrix} B_{11} & \cdots & B_{1d} \\ B_{21} & \ddots & \vdots \\ B_{k1} & \cdots & B_{kd} \end{bmatrix} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(n)} \\ | & | & & | \end{bmatrix}$$

$$= \underbrace{[A_1B_{11} + \cdots,\ A_1B_{12} + \cdots,\ \cdots,\ A_1B_{1d} + \cdots]}_{\in 1\times d} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(n)} \\ | & & | \end{bmatrix} \in d\times n$$

$$= [A_1B_{11}x_1^{(1)} + A_1B_{12}x_2^{(1)} \cdots + A_1B_{1d}x_d^{(1)},\ \cdots,\ \overset{x^{(n)}}{\downarrow}\ ]$$

$$\frac{df_W(x)}{dA_1} = \begin{bmatrix} B_{11}x_1^{(1)} + B_{12}x_2^{(1)} + \cdots B_{1d}x_d^{(1)} \\ B_{11}x_1^{(2)} + B_{12}x_2^{(2)} + \cdots B_{1d}x_d^{(2)} \\ \vdots \end{bmatrix} \in n\times 1$$

$$= \underset{n\times d}{X^T} \cdot \underset{d\times 1}{B_1^T}$$

$\Downarrow$

stack $A_1 \cdots A_k \Rightarrow \dfrac{df_W(x)}{dA} = X^T B^T \in n\times k.$

$$\Rightarrow A^{t+1} = A^t - \eta \nabla_A \mathcal{L}(W)$$

$$\mathcal{L}(W) = \frac{1}{2}(y - f_W(x))^2$$

$$\nabla_A \mathcal{L}(W) = -\underbrace{(y - f_W(x))}_{1\times n} \cdot \underbrace{\nabla_A f_W(x)}$$

$$\boxed{= (y - f_W(x)) \cdot X^T B^T}$$

Problem 1 B

Friday, January 27, 2023 11:55 AM

$$\underset{1\times n}{f_w(X)} = \underset{1\times k}{A}\,\underset{k\times d}{B}\,\underset{d\times n}{X}$$

$$\nabla_B \mathcal{L}(W) \in k\times d$$

$$B^{(t+1)} = B^{(t)} - \eta \cdot \nabla_B \mathcal{L}(W)$$

$$\mathcal{L}(W) = \frac{1}{2}\, \underbrace{\|\overset{1\times n}{y} - \overset{1\times n}{f_w(x)}\|_2^2}_{\in 1\times n} \in 1\times 1$$

$$= \frac{1}{2}\sum_{i=1}^{n}\Big(\underset{1\times 1}{y^{(i)}} - \underset{1\times 1}{f_w(x)^{(i)}}\Big)^2$$

$$\underset{k\times d}{\nabla_B \mathcal{L}(W)} = -\sum_{i=1}^{n} \Big(\underset{1\times 1}{y^{(i)} - f_w(x)^{(i)}}\Big)\, \underset{k\times d}{\nabla_B f_w(x)^{(i)}}$$

$$\nabla_{B_{ij}} f_w(x)^{(i)} \in 1\times 1$$

$$f_w(x)^{(i)} = f_w(x) = \underset{1\times k}{A}\,\underset{k\times d}{B}\,\underset{d\times 1}{x}$$

$$= \underbrace{[A_1 \cdots A_k] \begin{bmatrix} B_{11} & \cdots & B_{1d} \\ \vdots & B_{ij} & \\ B_{k1} & \cdots & B_{kd} \end{bmatrix}}_{1\times d} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

$i \le k \quad j \le d$

$$[\ldots\ \overset{j\text{th}}{A_i \cdot B_{ij}} + \cdots,\ \ldots\ ] \in 1\times d \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

$$\Rightarrow \boxed{\frac{d f_w(x)}{d B_{ij}} = A_i \cdot x_j}$$

$A \in 1\times k \quad X \in d\times n$

$A^T \in k\times 1 = k$

$X^T \in n\times d$

$$\underset{k\times 1}{A^T} \cdot \underset{1\times n}{(y - f_w(x))} \cdot \underset{n\times d}{X^T} \rightarrow k\times d$$

$$\boxed{\therefore\ B^{(t+1)} = B^{(t)} + \eta\, A^T (y - f_w(x)) \cdot X^T}$$

Problem 2

(a) Check CIFAR10

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [2]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

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

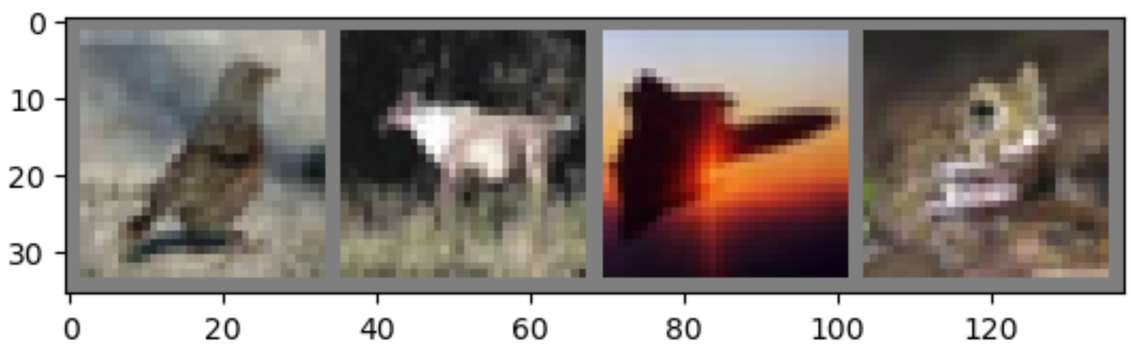
In [3]:
```
trainset
```

Out[3]:
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

In [4]:
```
testset
```

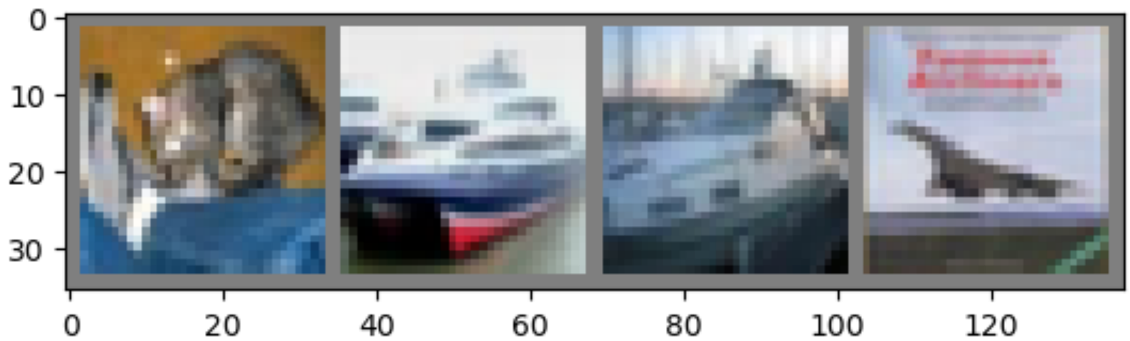
Out[4]:
```
Dataset CIFAR10
    Number of datapoints: 10000
    Root location: ./data
    Split: Test
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

In [5]:
```python
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```




```
bird  deer  plane frog
```

In [6]:
```python
# get some random training images
dataiter = iter(testloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```




```
cat   ship  ship  plane
```

In [7]:
```
images, labels = next(dataiter)
images.shape # 4 is batch size
```

Out[7]:
```
torch.Size([4, 3, 32, 32])
```

(b) Train 1 hidden layer ReLU

In [8]:
```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self, k=128):
        super(Net, self).__init__()
        self.k = k
        self.fc1 = nn.Linear(3 * 32 * 32, self.k, bias=False)
        self.fc2 = nn.Linear(self.k, 10, bias=False)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32) # [batchsize, 3072]
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = Net(k=128)
```

In [9]:
```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [11]:
```
# train the model
num_epochs = 20
train_loss_history = []
train_acc_history = []
test_loss_history = []
test_acc_history = []

# Loop through the number of epochs
for epoch in range(num_epochs):
```

```
    train_loss = 0.0
    train_acc = 0.0
    test_loss = 0.0
    test_acc = 0.0

    # set model to train mode
    model.train()
    # iterate over the training data
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(inputs)

        #compute the loss
        one_hot_labels = torch.nn.functional.one_hot(labels, num_classes=10)
        loss = criterion(outputs, one_hot_labels.float())
        loss.backward()
        optimizer.step()
        # increment the running loss and accuracy
        train_loss += loss.item()
        train_acc += (outputs.argmax(1) == labels).sum().item()

    # calculate the average training loss and accuracy
    train_loss /= len(trainloader)
    train_loss_history.append(train_loss)
    train_acc /= len(trainloader.dataset)
    train_acc_history.append(train_acc)

    # set the model to evaluation mode
    model.eval()
    with torch.no_grad():
        for inputs, labels in testloader:
            outputs = model(inputs)

            #compute the loss
            one_hot_labels = torch.nn.functional.one_hot(labels, num_classes
            loss = criterion(outputs, one_hot_labels.float())

            test_loss += loss.item()
            test_acc += (outputs.argmax(1) == labels).sum().item()

    # calculate the average validation loss and accuracy
    test_loss /= len(testloader)
    test_loss_history.append(test_loss)
    test_acc /= len(testloader.dataset)
    test_acc_history.append(test_acc)

    print(f'Epoch {epoch+1}/{num_epochs}, train loss: {train_loss:.4f}, trai
```

```
Epoch 1/20, train loss: 0.0944, train acc: 0.3199, val loss: 0.0879, val acc
: 0.3390
Epoch 2/20, train loss: 0.0952, train acc: 0.3307, val loss: 0.1536, val acc
: 0.2732
Epoch 3/20, train loss: 0.0942, train acc: 0.3431, val loss: 0.0938, val acc
: 0.3313
Epoch 4/20, train loss: 0.0941, train acc: 0.3450, val loss: 0.0846, val acc
: 0.3601
Epoch 5/20, train loss: 0.0935, train acc: 0.3508, val loss: 0.0916, val acc
: 0.3296
Epoch 6/20, train loss: 0.0951, train acc: 0.3520, val loss: 0.0854, val acc
: 0.3571
Epoch 7/20, train loss: 0.0928, train acc: 0.3552, val loss: 0.0904, val acc
: 0.3580
Epoch 8/20, train loss: 0.0945, train acc: 0.3527, val loss: 0.0873, val acc
: 0.3521
Epoch 9/20, train loss: 0.0949, train acc: 0.3622, val loss: 0.0998, val acc
: 0.3059
Epoch 10/20, train loss: 0.0947, train acc: 0.3583, val loss: 0.1001, val ac
c: 0.3121
Epoch 11/20, train loss: 0.0986, train acc: 0.3668, val loss: 0.0846, val ac
c: 0.3774
Epoch 12/20, train loss: 0.0927, train acc: 0.3628, val loss: 0.0929, val ac
c: 0.3458
Epoch 13/20, train loss: 0.0931, train acc: 0.3605, val loss: 0.0930, val ac
c: 0.3446
Epoch 14/20, train loss: 0.0931, train acc: 0.3635, val loss: 0.0958, val ac
c: 0.3356
Epoch 15/20, train loss: 0.0937, train acc: 0.3659, val loss: 0.1190, val ac
c: 0.2893
Epoch 16/20, train loss: 0.0925, train acc: 0.3662, val loss: 0.0881, val ac
c: 0.3417
Epoch 17/20, train loss: 0.0927, train acc: 0.3684, val loss: 0.1048, val ac
c: 0.3247
Epoch 18/20, train loss: 0.0927, train acc: 0.3687, val loss: 0.0908, val ac
c: 0.3416
Epoch 19/20, train loss: 0.0929, train acc: 0.3686, val loss: 0.1023, val ac
c: 0.3099
Epoch 20/20, train loss: 0.0923, train acc: 0.3713, val loss: 0.0980, val ac
c: 0.3216
```

After 20 epochs of training, it achieves 32% of accuracy on test set.

Problem 3

In [1]:
```
import torch
import numpy as np
```

In [2]:
```
torch.manual_seed(2139)
z = torch.normal(0, 1 , size=(100, 100), requires_grad=True)
```

In [3]:
```
z.shape
```

Out[3]:
```
torch.Size([100, 100])
```

In [4]:
```
k = 50
```

(a)

In [5]:
```
A = torch.normal(0, 0 , size=(100, k), requires_grad=True)
B = torch.normal(0, 0 , size=(k, 100), requires_grad=True)
```

In [6]:
```
def f(z):
    return A @ B @ z
```

In [7]:
```
 # evaluating data points with Mean Square Error (MSE)
def L(z, fz):
    diff = z - fz
    return 0.5 * (torch.norm(diff, p=2)**2)
```

In [8]:
```
steps = 10
lr = 1e-5

def train(steps, lr, A, B):
    losses = []
    for i in range(steps):
        # Generate Prediction
        fz = f(z)
        # Get the loss and perform backpropagation
        loss = L(z, fz)
        losses.append(loss)
        loss.backward() # get gradient
        # Let's update the weights
        with torch.no_grad():
            A -= lr * A.grad
            B -= lr * B.grad
            # Set the gradients to zero
            A.grad.zero_()
            B.grad.zero_()
#          print(f"step {i}: Loss: {loss}")
    print(f"A==0: ", torch.all(A==0))
    print(f"B==0: ", torch.all(B==0))
    print(f"minimal loss achieved: {min(losses)}")
```

In [9]:
```
train(steps, lr, A, B)
```

```
A==0:  tensor(True)
B==0:  tensor(True)
minimal loss achieved: 4919.1142578125
```

Weights after training is always 0 - because $L(w)$ is always 0 thus results in gradients of 0 which means no update.

(b)

In [10]:
```
A = torch.normal(0, 1/k , size=(100, k), requires_grad=True)
B = torch.normal(0, 1/k , size=(k, 100), requires_grad=True)
```

In [11]:
```
steps = 1000
lrs = [1e-4, 1e-3, 1e-2, 1e-1]

for lr in lrs:
    print("lr: ", lr)
    train(steps, lr, A, B)
```

```
lr:  0.0001
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 518.71435546875
lr:  0.001
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 506.402587890625
lr:  0.01
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 506.3055419921875
lr:  0.1
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: nan
```

The smallest training error achieved is 506.305 with the learning rate 0.01.

(c) CIFAR10 (optional)

In [12]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [13]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 1

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

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

In [14]:
```
# get some random training images
dataiter = iter(testloader)
images, labels = next(dataiter)

images.shape
```

Out[14]:
```
torch.Size([1, 3, 32, 32])
```

In [15]:
```
from tqdm import tqdm
import numpy as np

k=50
img = 3*32*32
A = torch.normal(0, 1/k , size=(img, k), requires_grad=True)
B = torch.normal(0, 1/k , size=(k, img), requires_grad=True)
lr = 1e-4
epoch = 10

def f(z):
    return A @ B @ z

def train(steps, lr, A, B):
    for e in range(epoch):
        losses = []
        print("epoch: ", e)
        for inputs, labels in tqdm(trainloader):
            z = inputs.view(3*32*32)
            # Generate Prediction
            fz = f(z)
            # Get the loss and perform backpropagation
            loss = L(z, fz)
#             print(loss)
            losses.append(loss)
            loss.backward() # get gradient
            # Let's update the weights
            with torch.no_grad():
                A -= lr * A.grad
                B -= lr * B.grad
                # Set the gradients to zero
                A.grad.zero_()
                B.grad.zero_()
        print(f"epoch loss: {np.mean(losses)}")
```

In [16]:
```
train(steps, lr, A, B)
```

```
epoch:  0
  1%||                                        | 301/50000 [00:15<43:10, 19.19i
t/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [16], in <cell line: 1>()
----> 1 train(steps, lr, A, B)

Input In [15], in train(steps, lr, A, B)
     24 #             print(loss)
     25             losses.append(loss)
---> 26             loss.backward() # get gradient
     27             # Let's update the weights
     28             with torch.no_grad():

File ~/anaconda3/lib/python3.9/site-packages/torch/_tensor.py:488, in Tensor
.backward(self, gradient, retain_graph, create_graph, inputs)
    478 if has_torch_function_unary(self):
    479     return handle_torch_function(
    480         Tensor.backward,
    481         (self,),
   (...)
    486         inputs=inputs,
    487     )
--> 488 torch.autograd.backward(
    489     self, gradient, retain_graph, create_graph, inputs=inputs
    490 )

File ~/anaconda3/lib/python3.9/site-packages/torch/autograd/__init__.py:197,
in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variable
s, inputs)
    192     retain_graph = create_graph
    194 # The reason we repeat same the comment below is that
    195 # some Python versions print out the first line of a multi-line func
tion
    196 # calls in the traceback and some print out the last line
--> 197 Variable._execution_engine.run_backward(  # Calls into the C++ engin
e to run the backward pass
    198     tensors, grad_tensors_, retain_graph, create_graph, inputs,
    199     allow_unreachable=True, accumulate_grad=True)

KeyboardInterrupt:
```

Sorry, it took so long time on cpu so give up at this point..

Problem 4

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

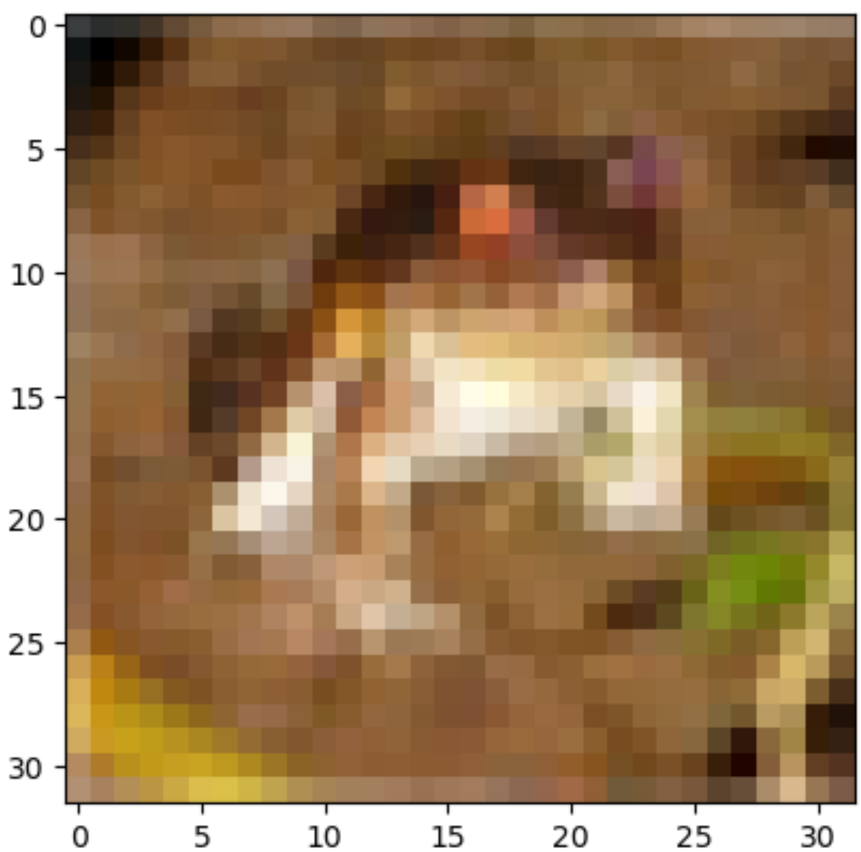
In [2]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 1

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=False, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

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

In [3]:
```
trainset
```

Out[3]:
```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

In [4]:
```python
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
z1, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(z1))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



frog

(a) Train an image with an error less than 1e-5

In [5]:
```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self, k):
        super(Net, self).__init__()
        self.k = k
        self.B = nn.Linear(3 * 32 * 32, self.k, bias=False)
        self.A = nn.Linear(self.k, 3 * 32 * 32, bias=False)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32) # [batchsize, 3072]
        x = self.B(x)
        x = F.relu(x)
        x = self.A(x)
        return x

f = Net(k=1024)
```

In [6]:
```
# evaluating data points with Mean Square Error (MSE)
def L(x, fx):
    x = x.view(-1, 3 * 32 * 32)
    diff = x - fx
    return 0.5 * (torch.norm(diff, p=2)**2)
```

In [7]:
```
steps = 100
lr = 1e-3

def train(steps, lr):
    losses = []
    for i in range(steps):
        # Generate Prediction
        fx = f(z1)
        # Get the loss and perform backpropagation
        loss = L(z1, fx)
        losses.append(loss)
        loss.backward() # get gradient
        # Let's update the weights
        with torch.no_grad():
            f.A.weight -= lr * f.A.weight.grad
            f.B.weight -= lr * f.B.weight.grad
            # Set the gradients to zero
            f.A.weight.grad.zero_()
            f.B.weight.grad.zero_()
        print(f"step {i}: Loss: {loss}")
    print(f"minimal loss achieved: {min(losses)}")
```

In [8]:
```
train(steps, lr)
```

```
step 0: Loss: 325.329833984375
step 1: Loss: 273.7133483886719
```

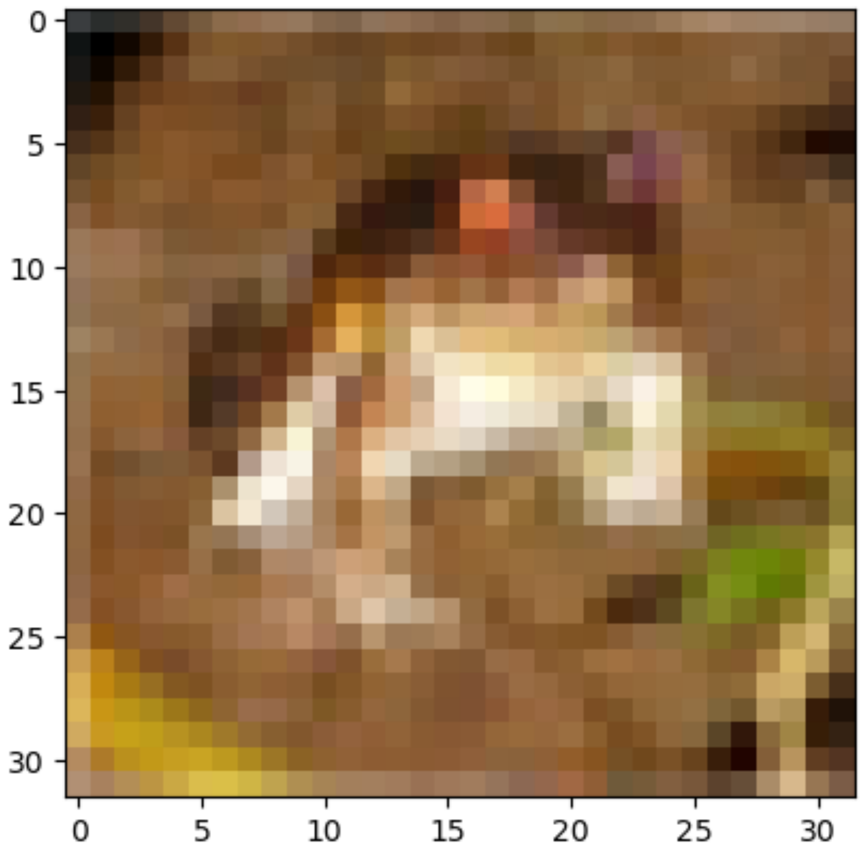
```
step 2: Loss: 244.14212036132812
step 3: Loss: 198.89141845703125
step 4: Loss: 139.93614196777344
step 5: Loss: 81.38546752929688
step 6: Loss: 39.56410598754883
step 7: Loss: 17.088428497314453
step 8: Loss: 6.952592849731445
step 9: Loss: 2.7564098834991455
step 10: Loss: 1.081404447555542
step 11: Loss: 0.4225172996520996
step 12: Loss: 0.1648177206516266
step 13: Loss: 0.06425297260284424
step 14: Loss: 0.025042491033673286
step 15: Loss: 0.009759376756846905
step 16: Loss: 0.003803201951086521
step 17: Loss: 0.0014820826472714543
step 18: Loss: 0.0005775515455752611
step 19: Loss: 0.00022506603272631764
step 20: Loss: 8.770507702138275e-05
step 21: Loss: 3.41781233146321e-05
step 22: Loss: 1.3318700439413078e-05
step 23: Loss: 5.19000332133146e-06
step 24: Loss: 2.0226091237418586e-06
step 25: Loss: 7.881436658863095e-07
step 26: Loss: 3.0718047128175385e-07
step 27: Loss: 1.196830083927125e-07
step 28: Loss: 4.663068509103141e-08
step 29: Loss: 1.8179873606527508e-08
step 30: Loss: 7.100092513923073e-09
step 31: Loss: 2.7733082497150008e-09
step 32: Loss: 1.0935854444227289e-09
step 33: Loss: 4.4747863747751637e-10
step 34: Loss: 2.041107144412635e-10
step 35: Loss: 1.1425811685672471e-10
step 36: Loss: 7.551961139773411e-11
step 37: Loss: 5.577729944583609e-11
step 38: Loss: 4.2543746303636e-11
step 39: Loss: 3.4071839100091594e-11
step 40: Loss: 2.7972523602981525e-11
step 41: Loss: 2.354040053165196e-11
step 42: Loss: 2.025936189642419e-11
step 43: Loss: 1.7818900868715737e-11
step 44: Loss: 1.579360080217196e-11
step 45: Loss: 1.4059483612050006e-11
step 46: Loss: 1.2719950960582427e-11
step 47: Loss: 1.1545031423920715e-11
step 48: Loss: 1.012597037469698e-11
step 49: Loss: 9.23511180722647e-12
step 50: Loss: 8.365031757551211e-12
step 51: Loss: 7.517048615512945e-12
step 52: Loss: 6.829037335620569e-12
step 53: Loss: 6.275987975840058e-12
step 54: Loss: 5.700450528278722e-12
```

```
step 55: Loss: 5.5734878517954556e-12
step 56: Loss: 5.199500986013961e-12
step 57: Loss: 4.980309567526042e-12
step 58: Loss: 4.499964637028064e-12
step 59: Loss: 4.109062048951451e-12
step 60: Loss: 3.924071137473284e-12
step 61: Loss: 3.685220531512989e-12
step 62: Loss: 3.554077388293053e-12
step 63: Loss: 3.2659242064220217e-12
step 64: Loss: 3.1987136463879073e-12
step 65: Loss: 3.0448976673369543e-12
step 66: Loss: 2.988852004434661e-12
step 67: Loss: 3.1237720743426678e-12
step 68: Loss: 2.5516674276998552e-12
step 69: Loss: 2.611591281773129e-12
step 70: Loss: 2.3714569804406116e-12
step 71: Loss: 2.5601135794639518e-12
step 72: Loss: 2.2941386203928493e-12
step 73: Loss: 2.2238042570593697e-12
step 74: Loss: 2.031854068676453e-12
step 75: Loss: 2.0095450910945223e-12
step 76: Loss: 2.1124907378933244e-12
step 77: Loss: 1.995121299072644e-12
step 78: Loss: 1.8691504507367673e-12
step 79: Loss: 1.846523151396995e-12
step 80: Loss: 1.812966443637265e-12
step 81: Loss: 1.6090878629526628e-12
step 82: Loss: 1.6250682440335784e-12
step 83: Loss: 1.5416921211874879e-12
step 84: Loss: 1.535273752746591e-12
step 85: Loss: 1.461442186885542e-12
step 86: Loss: 1.3970768986126814e-12
step 87: Loss: 1.3399627544694037e-12
step 88: Loss: 1.295652929562962 6e-12
step 89: Loss: 1.2132395782460392e-12
step 90: Loss: 1.2527132109418915e-12
step 91: Loss: 1.2355256791621305e-12
step 92: Loss: 1.1848438896677749e-12
step 93: Loss: 1.1214987272190058e-12
step 94: Loss: 1.084703615990279e-12
step 95: Loss: 1.192540857730684e-12
step 96: Loss: 1.0977885267493548e-12
step 97: Loss: 9.348216645221896e-13
step 98: Loss: 9.883848297101427e-13
step 99: Loss: 9.945187035009795e-13
minimal loss achieved: 9.348216645221896e-13
```

(b) Visualize

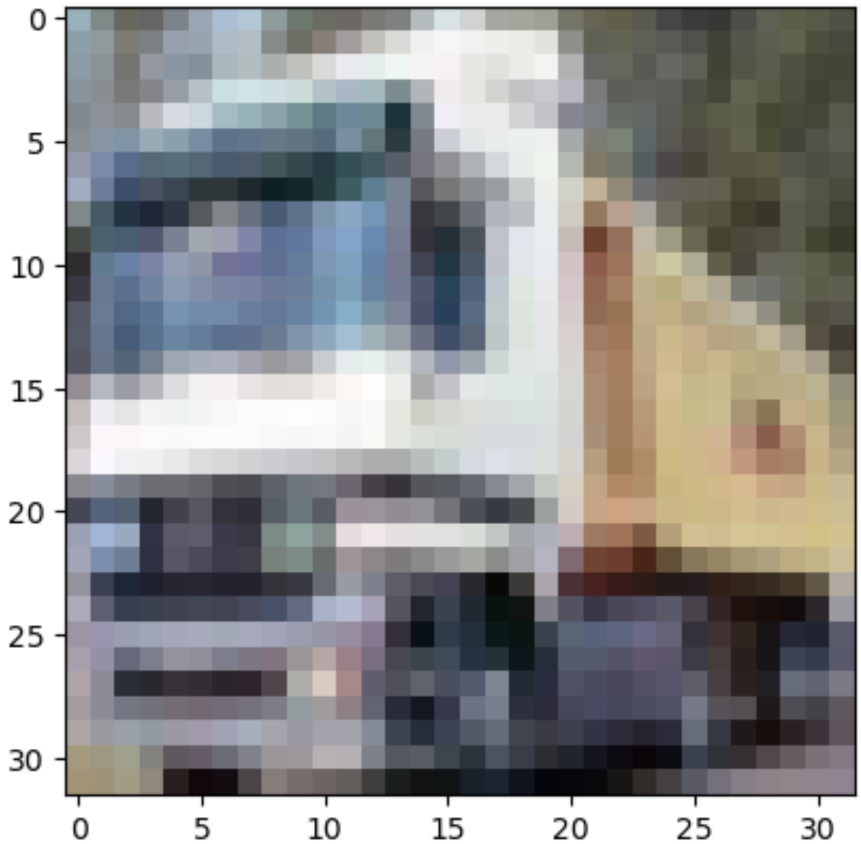
Training image

In [9]:
```
pred = f(z1)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



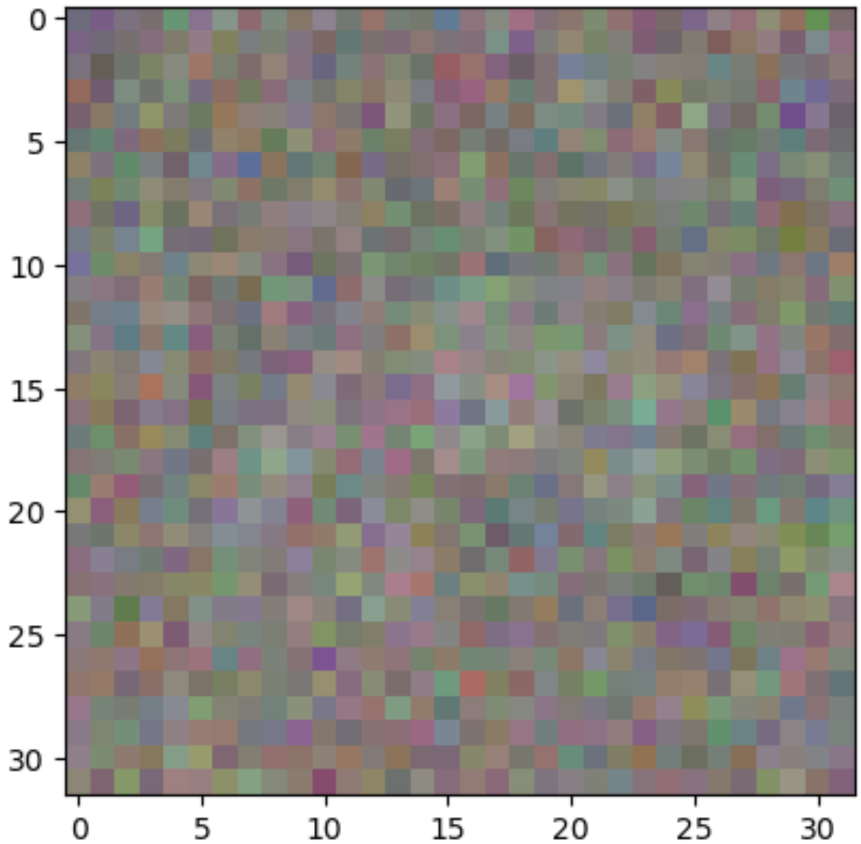
Random training image

In [10]:
```
# Random training image
random_train, labels = next(dataiter)
imshow(torchvision.utils.make_grid(random_train))
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



truck

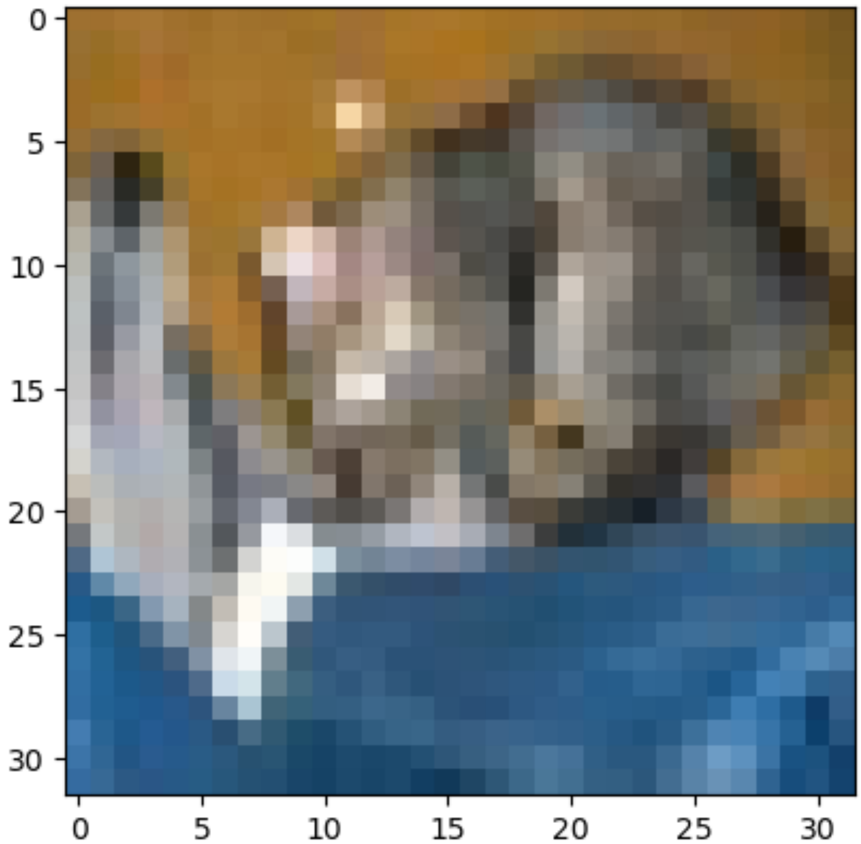
In [11]:
```
pred = f(random_train)
pred = pred.reshape_as(random_train)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



Random test image

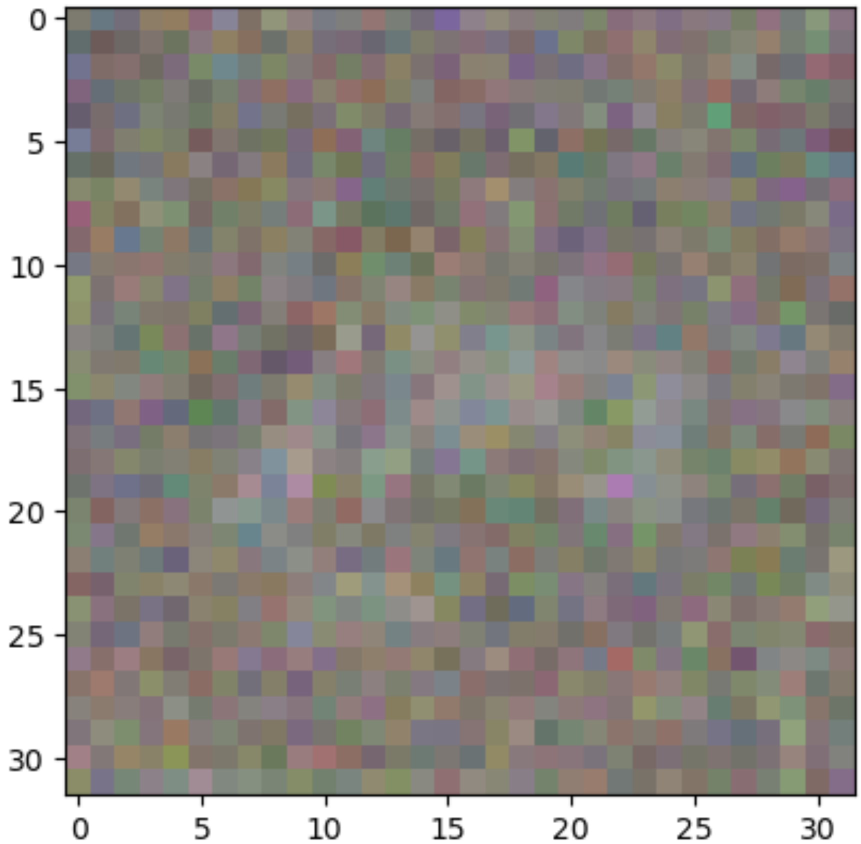
In [12]:

```
# get some random training images
dataiter = iter(testloader)
random_test, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(random_test))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



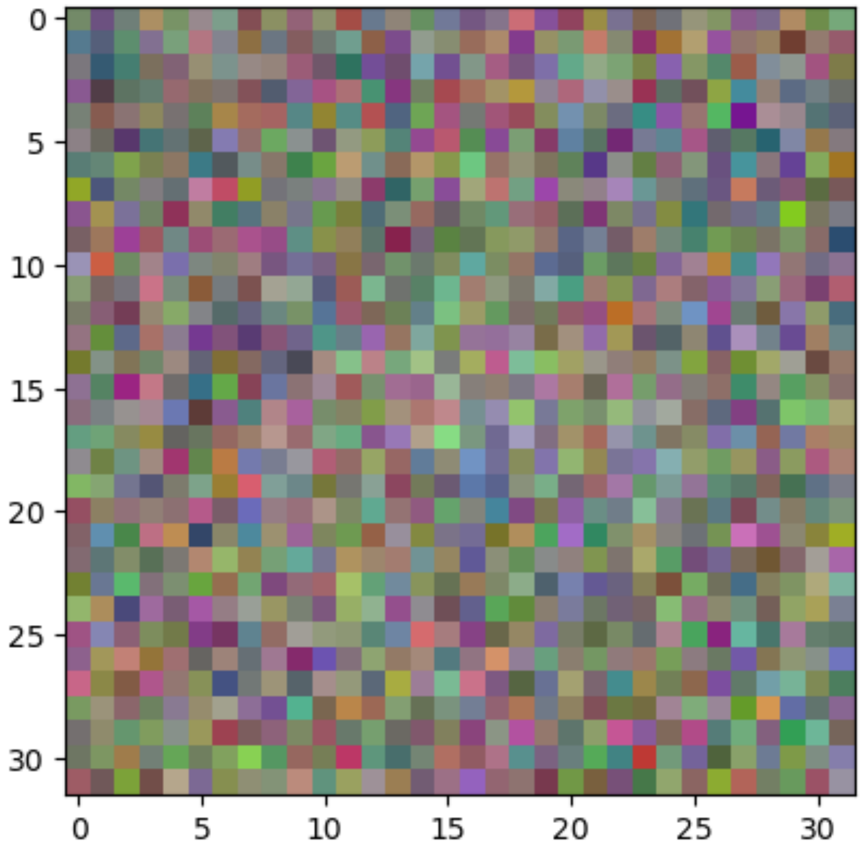
cat

In [13]:
```
pred = f(random_test)
pred = pred.reshape_as(random_test)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



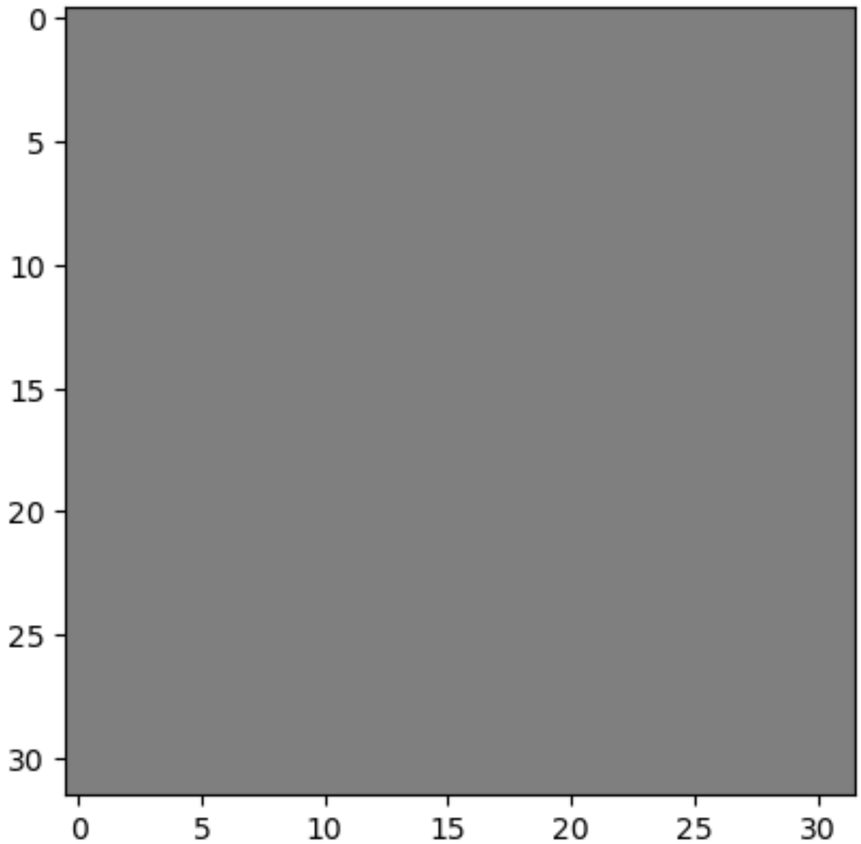
Random noise

In [14]:
```
noise = torch.normal(0, 1 , size=(1, 3, 32, 32), requires_grad=False)
pred = f(noise)
pred = pred.reshape_as(noise)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



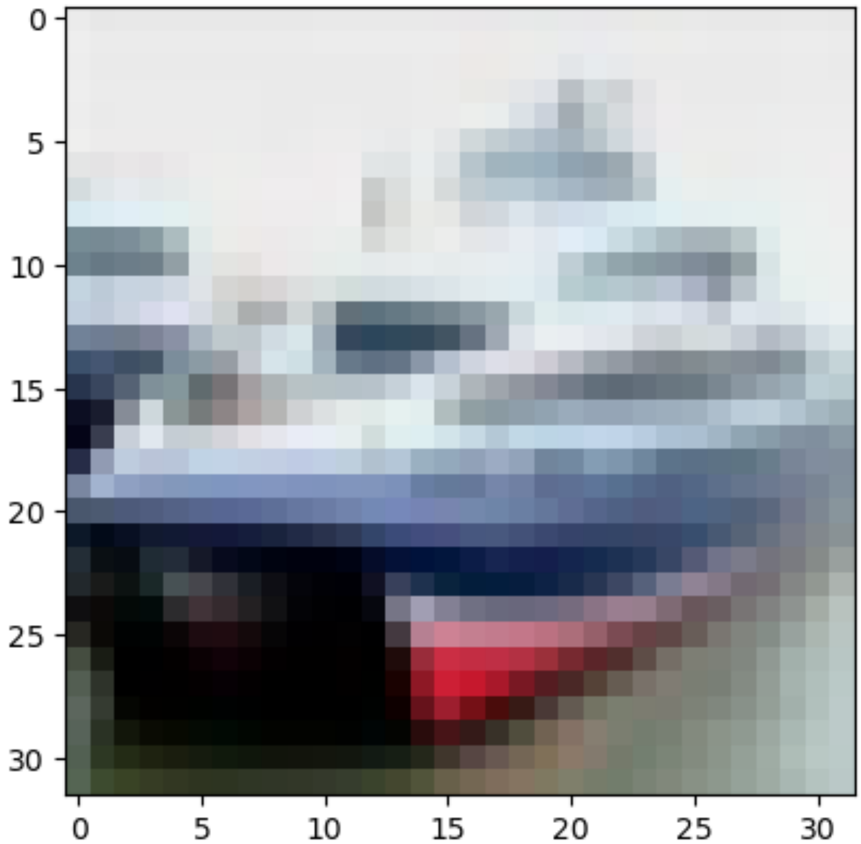
All zeros

In [15]:
```
zeros = torch.normal(0, 0 , size=(1, 3, 32, 32), requires_grad=False)
pred = f(zeros)
pred = pred.reshape_as(zeros)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



(c) Training on 2 images

In [16]:
```
z2, labels = next(dataiter)
# show images
imshow(torchvision.utils.make_grid(z2))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



ship

In [17]:
```
steps = 1000
lr = 1e-4
f = Net(k=1024)

def train2(steps, lr):
    losses = []
    for i in range(steps):
        if i % 2 == 0:
            x = z1
        else:
            x = z2
        # Generate Prediction
        fx = f(x)
        # Get the loss and perform backpropagation
        loss = L(x, fx)
        losses.append(loss)
        loss.backward() # get gradient
        # Let's update the weights
        with torch.no_grad():
            f.A.weight -= lr * f.A.weight.grad
            f.B.weight -= lr * f.B.weight.grad
            # Set the gradients to zero
            f.A.weight.grad.zero_()
            f.B.weight.grad.zero_()
        print(f"step {i}: Loss: {loss}")
    print(f"minimal loss achieved: {min(losses)}")
```

In [18]:
```
train2(steps, lr)
```

```
step 0: Loss: 327.88580322265625
step 1: Loss: 700.64794921875
step 2: Loss: 316.836669921875
step 3: Loss: 652.8576049804688
step 4: Loss: 308.36370849609375
step 5: Loss: 625.0802612304688
step 6: Loss: 301.4964904785156
step 7: Loss: 606.9642333984375
step 8: Loss: 295.74029541015625
step 9: Loss: 592.322998046875
step 10: Loss: 290.8751525878906
step 11: Loss: 578.8724365234375
step 12: Loss: 286.5518493652344
step 13: Loss: 565.206787109375
step 14: Loss: 282.59967041015625
step 15: Loss: 550.6947021484375
step 16: Loss: 278.98480224609375
step 17: Loss: 534.9141235351562
step 18: Loss: 275.56146240234375
step 19: Loss: 517.6426391601562
step 20: Loss: 272.2120361328125
step 21: Loss: 498.6922912597656
step 22: Loss: 268.87908935546875
```

```
step 23: Loss: 478.0023193359375
step 24: Loss: 265.52301025390625
step 25: Loss: 455.56219482421875
step 26: Loss: 262.07733154296875
step 27: Loss: 431.41168212890625
step 28: Loss: 258.5296936035156
step 29: Loss: 405.76983642578125
step 30: Loss: 254.87376403808594
step 31: Loss: 378.8576965332031
step 32: Loss: 251.06134033203125
step 33: Loss: 350.99676513671875
step 34: Loss: 247.087158203125
step 35: Loss: 322.5587158203125
step 36: Loss: 242.93609619140625
step 37: Loss: 293.9726867675781
step 38: Loss: 238.6042022705078
step 39: Loss: 265.6807861328125
step 40: Loss: 234.08609008789062
step 41: Loss: 238.1218719482422
step 42: Loss: 229.37242126464844
step 43: Loss: 211.70025634765625
step 44: Loss: 224.46142578125
step 45: Loss: 186.76055908203125
step 46: Loss: 219.35093688964844
step 47: Loss: 163.57046508789062
step 48: Loss: 214.0438995361328
step 49: Loss: 142.31459045410156
step 50: Loss: 208.54429626464844
step 51: Loss: 123.08268737792969
step 52: Loss: 202.8596954345703
step 53: Loss: 105.88780212402344
step 54: Loss: 196.99981689453125
step 55: Loss: 90.67615509033203
step 56: Loss: 190.97752380371094
step 57: Loss: 77.3438491821289
step 58: Loss: 184.80691528320312
step 59: Loss: 65.7522201538086
step 60: Loss: 178.5053253173828
step 61: Loss: 55.743553161621094
step 62: Loss: 172.09185791015625
step 63: Loss: 47.1514778137207
step 64: Loss: 165.58836364746094
step 65: Loss: 39.80912399291992
step 66: Loss: 159.01678466796875
step 67: Loss: 33.559146881103516
step 68: Loss: 152.40110778808594
step 69: Loss: 28.255844116210938
step 70: Loss: 145.76629638671875
step 71: Loss: 23.767404556274414
step 72: Loss: 139.13754272460938
step 73: Loss: 19.976667404174805
step 74: Loss: 132.54066467285156
step 75: Loss: 16.78044319152832
```

```
step 76: Loss: 126.0013656616211
step 77: Loss: 14.089128494262695
step 78: Loss: 119.54437255859375
step 79: Loss: 11.825434684753418
step 80: Loss: 113.19332122802734
step 81: Loss: 9.922957420349121
step 82: Loss: 106.97029113769531
step 83: Loss: 8.325211524963379
step 84: Loss: 100.89632415771484
step 85: Loss: 6.984189987182617
step 86: Loss: 94.99032592773438
step 87: Loss: 5.859060764312744
step 88: Loss: 89.26852416992188
step 89: Loss: 4.915434837341309
step 90: Loss: 83.74527740478516
step 91: Loss: 4.124213218688965
step 92: Loss: 78.43235778808594
step 93: Loss: 3.460897445678711
step 94: Loss: 73.3391342163086
step 95: Loss: 2.9048776626586914
step 96: Loss: 68.47264862060547
step 97: Loss: 2.4388275146484375
step 98: Loss: 63.83707046508789
step 99: Loss: 2.048197031021118
step 100: Loss: 59.43452072143555
step 101: Loss: 1.7207491397857666
step 102: Loss: 55.265079498291016
step 103: Loss: 1.446230411529541
step 104: Loss: 51.32695388793945
step 105: Loss: 1.2161104679107666
step 106: Loss: 47.61666488647461
step 107: Loss: 1.0231757164001465
step 108: Loss: 44.12925338745117
step 109: Loss: 0.8613846302032471
step 110: Loss: 40.85853958129883
step 111: Loss: 0.7256777286529541
step 112: Loss: 37.797306060791016
step 113: Loss: 0.6118183732032776
step 114: Loss: 34.93754959106445
step 115: Loss: 0.5162590742111206
step 116: Loss: 32.270687103271484
step 117: Loss: 0.43602922558784485
step 118: Loss: 29.787681579589844
step 119: Loss: 0.3686724305152893
step 120: Loss: 27.479257583618164
step 121: Loss: 0.31201332807540894
step 122: Loss: 25.335979461669922
step 123: Loss: 0.26440495252609253
step 124: Loss: 23.348468780517578
step 125: Loss: 0.22436116635799408
step 126: Loss: 21.507457733154297
step 127: Loss: 0.19064068794250488
step 128: Loss: 19.803810119628906
```

```
step 129: Loss: 0.16224084794521332
step 130: Loss: 18.228717803955078
step 131: Loss: 0.13830390572547913
step 132: Loss: 16.77366065979004
step 133: Loss: 0.11809547990560532
step 134: Loss: 15.430447578430176
step 135: Loss: 0.10103265196084976
step 136: Loss: 14.19129276275348
step 137: Loss: 0.08661586046218872
step 138: Loss: 13.048810005187988
step 139: Loss: 0.07439390569925308
step 140: Loss: 11.995981216430664
step 141: Loss: 0.0640515461564064
step 142: Loss: 11.026226043701172
step 143: Loss: 0.055261529982089996
step 144: Loss: 10.133353233337402
step 145: Loss: 0.04779597744345665
step 146: Loss: 9.311551094055176
step 147: Loss: 0.04144347086548805
step 148: Loss: 8.555465698242188
step 149: Loss: 0.036020323634147644
step 150: Loss: 7.859989166259766
step 151: Loss: 0.03139352798461914
step 152: Loss: 7.220425605773926
step 153: Loss: 0.02742904983460903
step 154: Loss: 6.632402420043945
step 155: Loss: 0.024029778316617012
step 156: Loss: 6.091865062713623
step 157: Loss: 0.02110857143998146
step 158: Loss: 5.595061302185059
step 159: Loss: 0.018592441454529762
step 160: Loss: 5.138516426086426
step 161: Loss: 0.01642162911593914
step 162: Loss: 4.719027996063232
step 163: Loss: 0.014540680684149265
step 164: Loss: 4.333626747131348
step 165: Loss: 0.01290850155055523
step 166: Loss: 3.979578971862793
step 167: Loss: 0.011488468386232853
step 168: Loss: 3.654357671737671
step 169: Loss: 0.010249658487737179
step 170: Loss: 3.355639934539795
step 171: Loss: 0.009166031144559383
step 172: Loss: 3.081282615661621
step 173: Loss: 0.00821552611887455
step 174: Loss: 2.829312801361084
step 175: Loss: 0.0073795076459646225
step 176: Loss: 2.597914457321167
step 177: Loss: 0.006642189808189869
step 178: Loss: 2.385416030883789
step 179: Loss: 0.005990123841911554
step 180: Loss: 2.190281391143799
step 181: Loss: 0.005411945749074221
```

```
step 182: Loss: 2.0110950469970703
step 183: Loss: 0.00489794509485364
step 184: Loss: 1.846558690071106
step 185: Loss: 0.004439824260771275
step 186: Loss: 1.6954782009124756
step 187: Loss: 0.004030512645840645
step 188: Loss: 1.5567545890808105
step 189: Loss: 0.0036639769095927477
step 190: Loss: 1.4293785095214844
step 191: Loss: 0.0033349506556987762
step 192: Loss: 1.3124237060546875
step 193: Loss: 0.0030390131287276745
step 194: Loss: 1.2050384283065796
step 195: Loss: 0.002772257197648287
step 196: Loss: 1.1064403057098389
step 197: Loss: 0.0025313773658126593
step 198: Loss: 1.015910267829895
step 199: Loss: 0.0023134632501751184
step 200: Loss: 0.9327893257141113
step 201: Loss: 0.0021159828174859285
step 202: Loss: 0.8564708828926086
step 203: Loss: 0.0019367544446140528
step 204: Loss: 0.78639817237854
step 205: Loss: 0.001773859723471105
step 206: Loss: 0.7220602035522461
step 207: Loss: 0.00162560457829386
step 208: Loss: 0.6629875302314758
step 209: Loss: 0.001490508671849966
step 210: Loss: 0.6087494492530823
step 211: Loss: 0.0013672763016074896
step 212: Loss: 0.5589501857757568
step 213: Loss: 0.001254746806807816
step 214: Loss: 0.5132259130477905
step 215: Loss: 0.0011518929386511445
step 216: Loss: 0.47124356031417847
step 217: Loss: 0.0010578109649941325
step 218: Loss: 0.43269675970077515
step 219: Loss: 0.0009716859785839915
step 220: Loss: 0.39730408787727356
step 221: Loss: 0.0008927848539315164
step 222: Loss: 0.3648072779178619 4
step 223: Loss: 0.0008204737096093595
step 224: Loss: 0.334969699382782
step 225: Loss: 0.0007541477680206299
step 226: Loss: 0.3075731694698334
step 227: Loss: 0.000693293462973088
step 228: Loss: 0.2824183702468872
step 229: Loss: 0.0006374387885443866
step 230: Loss: 0.2593213617801666
step 231: Loss: 0.0005861423560418189
step 232: Loss: 0.23811395466327667
step 233: Loss: 0.000539029308129102
step 234: Loss: 0.21864154934883118
```

```
step 235: Loss: 0.0004957331693731248
step 236: Loss: 0.20076194405555725
step 237: Loss: 0.0004559505614452064
step 238: Loss: 0.18434496223926544
step 239: Loss: 0.00041937301284633577
step 240: Loss: 0.169270858168602
step 241: Loss: 0.0003857449919451028
step 242: Loss: 0.1554296463727951
step 243: Loss: 0.00035482182283885777
step 244: Loss: 0.14272062480449677
step 245: Loss: 0.0003263828402850777
step 246: Loss: 0.13105101883411407
step 247: Loss: 0.0003002227458637208
step 248: Loss: 0.12033582478761673
step 249: Loss: 0.00027616112492978573
step 250: Loss: 0.11049695312976837
step 251: Loss: 0.00025402678875252604
step 252: Loss: 0.1014627143740654
step 253: Loss: 0.00023365739616565406
step 254: Loss: 0.093167245388031
step 255: Loss: 0.0002149244537577033
step 256: Loss: 0.0855502188205719
step 257: Loss: 0.0001976849016500637
step 258: Loss: 0.07855595648288727
step 259: Loss: 0.00018182701023761183
step 260: Loss: 0.07213369756937027
step 261: Loss: 0.00016723503358662128
step 262: Loss: 0.06623657792806625
step 263: Loss: 0.00015380996046587825
step 264: Loss: 0.06082165613770485
step 265: Loss: 0.00014145617024041712
step 266: Loss: 0.055849481374025345
step 267: Loss: 0.00013009064423386008
step 268: Loss: 0.05128379166126251
step 269: Loss: 0.00011963404540438205
step 270: Loss: 0.04709148034453392
step 271: Loss: 0.00011001431266777217
step 272: Loss: 0.04324183613061905
step 273: Loss: 0.00010116605699295178
step 274: Loss: 0.039707038551568985
step 275: Loss: 9.302308899350464e-05
step 276: Loss: 0.03646118938922882
step 277: Loss: 8.553446241421625e-05
step 278: Loss: 0.0334806852042675
step 279: Loss: 7.86422606324777e-05
step 280: Loss: 0.030743861570954323
step 281: Loss: 7.230581832118332e-05
step 282: Loss: 0.028230803087353706
step 283: Loss: 6.647563714068383e-05
step 284: Loss: 0.025923172011971474
step 285: Loss: 6.111565016908571e-05
step 286: Loss: 0.023804189637303352
step 287: Loss: 5.618337308987975e-05
```

```
step 288: Loss: 0.021858425810933113
step 289: Loss: 5.1648763474076986e-05
step 290: Loss: 0.020071713253855705
step 291: Loss: 4.7477511543547735e-05
step 292: Loss: 0.01843108795583248
step 293: Loss: 4.364215419627726e-05
step 294: Loss: 0.016924554482102394
step 295: Loss: 4.011435521533713e-05
step 296: Loss: 0.015541176311671734
step 297: Loss: 3.6871322663500905e-05
step 298: Loss: 0.014270894229412079
step 299: Loss: 3.388823461136781e-05
step 300: Loss: 0.013104432262480259
step 301: Loss: 3.1146275432547554e-05
step 302: Loss: 0.01203331258147955
step 303: Loss: 2.862545989046339e-05
step 304: Loss: 0.011049763299524784
step 305: Loss: 2.6307699954486452e-05
step 306: Loss: 0.010146606713533401
step 307: Loss: 2.417707582935691e-05
step 308: Loss: 0.009317274205386639
step 309: Loss: 2.221788417955395e-05
step 310: Loss: 0.008555716834962368
step 311: Loss: 2.0417122868821025e-05
step 312: Loss: 0.00785643607378006
step 313: Loss: 1.8762271793093532e-05
step 314: Loss: 0.007214294746518135
step 315: Loss: 1.723987952573225e-05
step 316: Loss: 0.006624647881835699
step 317: Loss: 1.5841640561120585e-05
step 318: Loss: 0.006083191838115454
step 319: Loss: 1.4556134374288376e-05
step 320: Loss: 0.005585992243140936
step 321: Loss: 1.3374574336921796e-05
step 322: Loss: 0.005129439290612936
step 323: Loss: 1.2289238839002792e-05
step 324: Loss: 0.0047101895324885845
step 325: Loss: 1.1290631846350152e-05
step 326: Loss: 0.004325216170400381
step 327: Loss: 1.0373551958764438e-05
step 328: Loss: 0.003971708007156849
step 329: Loss: 9.530929673928767e-06
step 330: Loss: 0.0036470878403633833
step 331: Loss: 8.755879207456019e-06
step 332: Loss: 0.0033490105997771025
step 333: Loss: 8.044251444516703e-06
step 334: Loss: 0.0030752921011298895
step 335: Loss: 7.390419796138303e-06
step 336: Loss: 0.002823940245434642
step 337: Loss: 6.7894970925408415e-06
step 338: Loss: 0.002593139884993434
step 339: Loss: 6.237150500965072e-06
step 340: Loss: 0.0023812006693333387
```

```
step 341: Loss: 5.729769782192307e-06
step 342: Loss: 0.002186582889407873
step 343: Loss: 5.263722869131016e-06
step 344: Loss: 0.002007870702072978
step 345: Loss: 4.83527082906221e-06
step 346: Loss: 0.0018437657272443175
step 347: Loss: 4.442124009074178e-06
step 348: Loss: 0.001693075057119131
step 349: Loss: 4.080366124981083e-06
step 350: Loss: 0.001554702641442418
step 351: Loss: 3.7482420793821802e-06
step 352: Loss: 0.001427631825208664
step 353: Loss: 3.4428344406478573e-06
step 354: Loss: 0.0013109489809721708
step 355: Loss: 3.1628246688342188e-06
step 356: Loss: 0.00120381114538759
step 357: Loss: 2.9054342576273484e-06
step 358: Loss: 0.0011054235510528088
step 359: Loss: 2.668485421963851e-06
step 360: Loss: 0.0010150778107345104
step 361: Loss: 2.451112550261314e-06
step 362: Loss: 0.0009321138495579362
step 363: Loss: 2.2512836039823014e-06
step 364: Loss: 0.00085593038238585
step 365: Loss: 2.068016556222574e-06
step 366: Loss: 0.0007859761826694012
step 367: Loss: 1.8994512629433302e-06
step 368: Loss: 0.0007217382080852985
step 369: Loss: 1.744724613672588e-06
step 370: Loss: 0.0006627531838603318
step 371: Loss: 1.602469069439394e-06
step 372: Loss: 0.0006085839122533798
step 373: Loss: 1.4720556009706343e-06
step 374: Loss: 0.0005588484345935285
step 375: Loss: 1.3521408845917904e-06
step 376: Loss: 0.0005131696816533804
step 377: Loss: 1.2420860002748668e-06
step 378: Loss: 0.00047122829710133374
step 379: Loss: 1.1407032616261858e-06
step 380: Loss: 0.00043271551840007305
step 381: Loss: 1.0477057230673381e-06
step 382: Loss: 0.00039734976598992944
step 383: Loss: 9.624332051316742e-07
step 384: Loss: 0.00036487303441390395
step 385: Loss: 8.839895713208534e-07
step 386: Loss: 0.00033505188184790313
step 387: Loss: 8.120248935483687e-07
step 388: Loss: 0.00030766858253628016
step 389: Loss: 7.458449999830918e-07
step 390: Loss: 0.0002825235715135932
step 391: Loss: 6.850228260191216e-07
step 392: Loss: 0.00025943282525986433
step 393: Loss: 6.292334546742495e-07
```

```
step 394: Loss: 0.0002382282546022907
step 395: Loss: 5.782278549304465e-07
step 396: Loss: 0.00021875939273741096
step 397: Loss: 5.310517963152961e-07
step 398: Loss: 0.0002008784795179963
step 399: Loss: 4.878539243691193e-07
step 400: Loss: 0.00018445984460413456
step 401: Loss: 4.4814794364356203e-07
step 402: Loss: 0.00016938372573349625
step 403: Loss: 4.1160089381264697e-07
step 404: Loss: 0.00015554130368400365
step 405: Loss: 3.782268436225422e-07
step 406: Loss: 0.00014282793563324958
step 407: Loss: 3.4747407084978477e-07
step 408: Loss: 0.00013115293404553086
step 409: Loss: 3.193229076714488e-07
step 410: Loss: 0.00012043327296851203
step 411: Loss: 2.932901281837985e-07
step 412: Loss: 0.0001105892370105721
step 413: Loss: 2.694440865980141e-07
step 414: Loss: 0.00010154980554943904
step 415: Loss: 2.4759501116022875e-07
step 416: Loss: 9.325049177277833e-05
step 417: Loss: 2.2752413997295662e-07
step 418: Loss: 8.562628499930725e-05
step 419: Loss: 2.0905596898046497e-07
step 420: Loss: 7.862808706704527e-05
step 421: Loss: 1.921175822872101e-07
step 422: Loss: 7.220054976642132e-05
step 423: Loss: 1.7665649920672877e-07
step 424: Loss: 6.629848212469369e-05
step 425: Loss: 1.6236306521477673e-07
step 426: Loss: 6.08779264439363e-05
step 427: Loss: 1.4925493019291025e-07
step 428: Loss: 5.5902353778947145e-05
step 429: Loss: 1.3721214031647833e-07
step 430: Loss: 5.133172089699656e-05
step 431: Loss: 1.2615964806172997e-07
step 432: Loss: 4.713556700153276e-05
step 433: Loss: 1.1608423022835268e-07
step 434: Loss: 4.328175782575272e-05
step 435: Loss: 1.0676558304112405e-07
step 436: Loss: 3.974303399445489e-05
step 437: Loss: 9.821942370535908e-08
step 438: Loss: 3.649448626674712e-05
step 439: Loss: 9.033868053620608e-08
step 440: Loss: 3.350940460222773e-05
step 441: Loss: 8.322024314111331e-08
step 442: Loss: 3.076986104133539e-05
step 443: Loss: 7.66110872518766e-08
step 444: Loss: 2.8253693017177284e-05
step 445: Loss: 7.057583673031331e-08
step 446: Loss: 2.594357829366345e-05
```

```
step 447: Loss: 6.506156324803669e-08
step 448: Loss: 2.3821761715225875e-05
step 449: Loss: 5.995127594360383e-08
step 450: Loss: 2.187398240494076e-05
step 451: Loss: 5.530831970190775e-08
step 452: Loss: 2.0084669813513756e-05
step 453: Loss: 5.103007794104997e-08
step 454: Loss: 1.8442202417645603e-05
step 455: Loss: 4.711765910769827e-08
step 456: Loss: 1.693348167464137e-05
step 457: Loss: 4.350940940867076e-08
step 458: Loss: 1.5548606825177558e-05
step 459: Loss: 4.021745425575318e-08
step 460: Loss: 1.4276944057201035e-05
step 461: Loss: 3.721442354276405e-08
step 462: Loss: 1.3108860912325326e-05
step 463: Loss: 3.447174989901214e-08
step 464: Loss: 1.203664123750059e-05
step 465: Loss: 3.190750419435062e-08
step 466: Loss: 1.1052125046262518e-05
step 467: Loss: 2.962218914603909e-08
step 468: Loss: 1.0147947250516154e-05
step 469: Loss: 2.7493687326796135e-08
step 470: Loss: 9.317822332377546e-06
step 471: Loss: 2.5556525784509176e-08
step 472: Loss: 8.555300155421719e-06
step 473: Loss: 2.3768393475620542e-08
step 474: Loss: 7.855060175643303e-06
step 475: Loss: 2.2159898804829936e-08
step 476: Loss: 7.212388936750358e-06
step 477: Loss: 2.0673804002058205e-08
step 478: Loss: 6.622700766456546e-06
step 479: Loss: 1.9323568523077483e-08
step 480: Loss: 6.080318598833401e-06
step 481: Loss: 1.805579330493856e-08
step 482: Loss: 5.583051915891701e-06
step 483: Loss: 1.6923054957374006e-08
step 484: Loss: 5.126212272443809e-06
step 485: Loss: 1.587434894645412e-08
step 486: Loss: 4.70667919216794e-06
step 487: Loss: 1.4915473300902704e-08
step 488: Loss: 4.321507276472403e-06
step 489: Loss: 1.4041192208935627e-08
step 490: Loss: 3.967792963521788e-06
step 491: Loss: 1.3205747606548357e-08
step 492: Loss: 3.6432666092878208e-06
step 493: Loss: 1.2472930244200597e-08
step 494: Loss: 3.3452145089540863e-06
step 495: Loss: 1.1789081710844584e-08
step 496: Loss: 3.0713215437572217e-06
step 497: Loss: 1.116304204629159 4e-08
step 498: Loss: 2.8200911401654594e-06
step 499: Loss: 1.056677856325905e-08
```

```
step 500: Loss: 2.5892429675877793e-06
step 501: Loss: 1.0018290197422175e-08
step 502: Loss: 2.377312284806976e-06
step 503: Loss: 9.53961443173057e-09
step 504: Loss: 2.182853677368257e-06
step 505: Loss: 9.077125717738e-09
step 506: Loss: 2.0042516553075984e-06
step 507: Loss: 8.648796345767096e-09
step 508: Loss: 1.8401573242954328e-06
step 509: Loss: 8.257265982081208e-09
step 510: Loss: 1.6896292436285876e-06
step 511: Loss: 7.886716169025476e-09
step 512: Loss: 1.5514407323280466e-06
step 513: Loss: 7.546471003649913e-09
step 514: Loss: 1.4245255215428188e-06
step 515: Loss: 7.22378823425629e-09
step 516: Loss: 1.3079825293971226e-06
step 517: Loss: 6.926044182620217e-09
step 518: Loss: 1.2009379588562297e-06
step 519: Loss: 6.6512995111622786e-09
step 520: Loss: 1.1027079835912446e-06
step 521: Loss: 6.380647121773109e-09
step 522: Loss: 1.0124625759999617e-06
step 523: Loss: 6.133018981557825e-09
step 524: Loss: 9.297144174524874e-07
step 525: Loss: 5.9004103825088805e-09
step 526: Loss: 8.537812163922354e-07
step 527: Loss: 5.674697600710488e-09
step 528: Loss: 7.83918721936061e-07
step 529: Loss: 5.464320995685057e-09
step 530: Loss: 7.197286322480068e-07
step 531: Loss: 5.263432356628073e-09
step 532: Loss: 6.609417368963477e-07
step 533: Loss: 5.0766479908759266e-09
step 534: Loss: 6.06965841143392e-07
step 535: Loss: 4.88530327302783e-09
step 536: Loss: 5.574665919994004e-07
step 537: Loss: 4.7209294251615574e-09
step 538: Loss: 5.118938020132191e-07
step 539: Loss: 4.553989185751561e-09
step 540: Loss: 4.700858369233174e-07
step 541: Loss: 4.394036690058556e-09
step 542: Loss: 4.317437287681969e-07
step 543: Loss: 4.244768980754543e-09
step 544: Loss: 3.96550603909418e-07
step 545: Loss: 4.0954821756145066e-09
step 546: Loss: 3.641771115781012e-07
step 547: Loss: 3.955983540748775e-09
step 548: Loss: 3.345535333210137e-07
step 549: Loss: 3.8167859983673225e-09
step 550: Loss: 3.0727190392099146e-07
step 551: Loss: 3.6925837942902717e-09
step 552: Loss: 2.8231519877408573e-07
```

```
step 553: Loss: 3.565914896697109e-09
step 554: Loss: 2.5940531145351997e-07
step 555: Loss: 3.4480849286921966e-09
step 556: Loss: 2.383656010351842 3e-07
step 557: Loss: 3.3194620385756934e-09
step 558: Loss: 2.1904072866618662e-07
step 559: Loss: 3.213282084857383e-09
step 560: Loss: 2.013361211083975e-07
step 561: Loss: 3.099040357668059e-09
step 562: Loss: 1.8512477595322707e-07
step 563: Loss: 3.001125792323478e-09
step 564: Loss: 1.701278762311631e-07
step 565: Loss: 2.9027020786998037e-09
step 566: Loss: 1.5653625951017602e-07
step 567: Loss: 2.8033155796691744e-09
step 568: Loss: 1.4402969839011348e-07
step 569: Loss: 2.7096320742714397e-09
step 570: Loss: 1.3251207064968185e-07
step 571: Loss: 2.617553285233498e-09
step 572: Loss: 1.2198032095511735e-07
step 573: Loss: 2.526536535540913e-09
step 574: Loss: 1.1229824536940214e-07
step 575: Loss: 2.4476358717606672e-09
step 576: Loss: 1.0344851375521102e-07
step 577: Loss: 2.364405782273593e-09
step 578: Loss: 9.532579525739493e-08
step 579: Loss: 2.286483002933437e-09
step 580: Loss: 8.790492245225323e-08
step 581: Loss: 2.2150505873952397e-09
step 582: Loss: 8.10585021326915e-08
step 583: Loss: 2.137052534934014e-09
step 584: Loss: 7.482163510985629e-08
step 585: Loss: 2.064820092684272e-09
step 586: Loss: 6.911259475828047e-08
step 587: Loss: 1.9961654551536867e-09
step 588: Loss: 6.389757345459657e-08
step 589: Loss: 1.9336001688685656e-09
step 590: Loss: 5.914560574638017e-08
step 591: Loss: 1.8720740513344936e-09
step 592: Loss: 5.4737203214472174e-08
step 593: Loss: 1.8095361875580807e-09
step 594: Loss: 5.073203013239436e-08
step 595: Loss: 1.746266020674625e-09
step 596: Loss: 4.7048587248355034e-08
step 597: Loss: 1.6920348455684575e-09
step 598: Loss: 4.366626882301716e-08
step 599: Loss: 1.634821167328937e-09
step 600: Loss: 4.0610181883948826e-08
step 601: Loss: 1.5826889798731258e-09
step 602: Loss: 3.7780658601604955e-08
step 603: Loss: 1.5328107672019087e-09
step 604: Loss: 3.5207385451485607e-08
step 605: Loss: 1.4853187568775184e-09
```

```
step 606: Loss: 3.283547655996699e-08
step 607: Loss: 1.4396807079819496e-09
step 608: Loss: 3.06704137642555e-08
step 609: Loss: 1.3902715645386365e-09
step 610: Loss: 2.8698847742703038e-08
step 611: Loss: 1.3512668761705982e-09
step 612: Loss: 2.6884622528200453e-08
step 613: Loss: 1.308384622866754e-09
step 614: Loss: 2.520530451022296e-08
step 615: Loss: 1.2706969920728284e-09
step 616: Loss: 2.3673813132063515e-08
step 617: Loss: 1.2337010302232443e-09
step 618: Loss: 2.2276656963526875e-08
step 619: Loss: 1.1933998234070486e-09
step 620: Loss: 2.098066431699408e-08
step 621: Loss: 1.156175932770509e-09
step 622: Loss: 1.9787272265148204e-08
step 623: Loss: 1.1212651918057759e-09
step 624: Loss: 1.870595234265693e-08
step 625: Loss: 1.091030710220764e-09
step 626: Loss: 1.769720547883935e-08
step 627: Loss: 1.0585522458583796e-09
step 628: Loss: 1.6764696297855153e-08
step 629: Loss: 1.0289905594262905e-09
step 630: Loss: 1.5917935414222484e-08
step 631: Loss: 1.001406735312571e-09
step 632: Loss: 1.51274353044073e-08
step 633: Loss: 9.748906126816337e-10
step 634: Loss: 1.4382494306630633e-08
step 635: Loss: 9.48544354173464e-10
step 636: Loss: 1.3701424883549862e-08
step 637: Loss: 9.215090912562118e-10
step 638: Loss: 1.3069886506400508e-08
step 639: Loss: 8.981169141719647e-10
step 640: Loss: 1.2497723744786526e-08
step 641: Loss: 8.720829058894708e-10
step 642: Loss: 1.1949754075146757e-08
step 643: Loss: 8.478653334975661e-10
step 644: Loss: 1.1443885838957613e-08
step 645: Loss: 8.259004591337771e-10
step 646: Loss: 1.0975965025750156e-08
step 647: Loss: 8.033388954054033e-10
step 648: Loss: 1.0527610783128694e-08
step 649: Loss: 7.817927416553516e-10
step 650: Loss: 1.0103111236503537e-08
step 651: Loss: 7.627348197480899e-10
step 652: Loss: 9.716238480450556e-09
step 653: Loss: 7.438095694922708e-10
step 654: Loss: 9.345063389787356e-09
step 655: Loss: 7.278335156790661e-10
step 656: Loss: 9.007439238928328e-09
step 657: Loss: 7.097888943263797e-10
step 658: Loss: 8.68367333595188e-09
```

```
step 659: Loss: 6.929546936262909e-10
step 660: Loss: 8.377000426662562e-09
step 661: Loss: 6.764513393875404e-10
step 662: Loss: 8.088358427471576e-09
step 663: Loss: 6.597372093075649e-10
step 664: Loss: 7.811876479024704e-09
step 665: Loss: 6.453075296342092e-10
step 666: Loss: 7.550577940662606e-09
step 667: Loss: 6.322387613444391e-10
step 668: Loss: 7.316441674731777e-09
step 669: Loss: 6.185546519432705e-10
step 670: Loss: 7.0736776436319815e-09
step 671: Loss: 6.036963706712584e-10
step 672: Loss: 6.850993994333976e-09
step 673: Loss: 5.909984168717131e-10
step 674: Loss: 6.642418171054487e-09
step 675: Loss: 5.795548485565405e-10
step 676: Loss: 6.430780796762292e-09
step 677: Loss: 5.678383319107638e-10
step 678: Loss: 6.239929906115549e-09
step 679: Loss: 5.556227700154182e-10
step 680: Loss: 6.060871360347164e-09
step 681: Loss: 5.441612715983979e-10
step 682: Loss: 5.884166043301775e-09
step 683: Loss: 5.323366192300227e-10
step 684: Loss: 5.715433015751614e-09
step 685: Loss: 5.215494147670086e-10
step 686: Loss: 5.556566762265902e-09
step 687: Loss: 5.113838796866332e-10
step 688: Loss: 5.403820058091924e-09
step 689: Loss: 5.009138104306032e-10
step 690: Loss: 5.260434310372375e-09
step 691: Loss: 4.904216477363832e-10
step 692: Loss: 5.1260515832041165e-09
step 693: Loss: 4.799644570674388e-10
step 694: Loss: 4.988945701001057e-09
step 695: Loss: 4.706138811982896e-10
step 696: Loss: 4.860537305972912e-09
step 697: Loss: 4.610543336003303e-10
step 698: Loss: 4.739659331676194e-09
step 699: Loss: 4.5348474975170916e-10
step 700: Loss: 4.625064331520434e-09
step 701: Loss: 4.4431339163431005e-10
step 702: Loss: 4.508954543069876e-09
step 703: Loss: 4.364367756082288e-10
step 704: Loss: 4.3998342746931485e-09
step 705: Loss: 4.2864023441779864e-10
step 706: Loss: 4.297135980380062e-09
step 707: Loss: 4.2113348919237126e-10
step 708: Loss: 4.199149028494276e-09
step 709: Loss: 4.1312925302960934e-10
step 710: Loss: 4.1059671218590665e-09
step 711: Loss: 4.0468442485952494e-10
```

```
step 712: Loss: 4.0111944876741745e-09
step 713: Loss: 3.9738526358412685e-10
step 714: Loss: 3.920797020384725e-09
step 715: Loss: 3.8968725468713217e-10
step 716: Loss: 3.8338399122039846e-09
step 717: Loss: 3.832988371144097e-10
step 718: Loss: 3.751178034860914e-09
step 719: Loss: 3.7739347757970165e-10
step 720: Loss: 3.6712408668648777e-09
step 721: Loss: 3.700315609478366e-10
step 722: Loss: 3.5973415357659633e-09
step 723: Loss: 3.639488155293691e-10
step 724: Loss: 3.523079827871811e-09
step 725: Loss: 3.5827737998594955e-10
step 726: Loss: 3.4489147093808015e-09
step 727: Loss: 3.520150559932489e-10
step 728: Loss: 3.3775762187104874e-09
step 729: Loss: 3.459126873828211e-10
step 730: Loss: 3.309648333171822e-09
step 731: Loss: 3.406254722726487e-10
step 732: Loss: 3.242547563786502e-09
step 733: Loss: 3.359446609785266e-10
step 734: Loss: 3.1786715481985084e-09
step 735: Loss: 3.307951967901346e-10
step 736: Loss: 3.115644631179748e-09
step 737: Loss: 3.257223102348661e-10
step 738: Loss: 3.05509928466563e-09
step 739: Loss: 3.2077118738982335e-10
step 740: Loss: 2.9967237580308392e-09
step 741: Loss: 3.151584826444065e-10
step 742: Loss: 2.938932652796211e-09
step 743: Loss: 3.095491363236391e-10
step 744: Loss: 2.878831617536548e-09
step 745: Loss: 3.038793383591809e-10
step 746: Loss: 2.825199185707561e-09
step 747: Loss: 2.998048198588066e-10
step 748: Loss: 2.773005380873883e-09
step 749: Loss: 2.95376417769333e-10
step 750: Loss: 2.7206068509144643e-09
step 751: Loss: 2.9115179711602934e-10
step 752: Loss: 2.6755866411320994e-09
step 753: Loss: 2.871828885808725e-10
step 754: Loss: 2.6238706762882202e-09
step 755: Loss: 2.829842749019207e-10
step 756: Loss: 2.5793627234094174e-09
step 757: Loss: 2.7932511859063425e-10
step 758: Loss: 2.535659682223468e-09
step 759: Loss: 2.756295192085645e-10
step 760: Loss: 2.4893842542894618e-09
step 761: Loss: 2.725680792181606e-10
step 762: Loss: 2.4458082226175293e-09
step 763: Loss: 2.6944887987490063e-10
step 764: Loss: 2.4018502742251258e-09
```

```
step 765: Loss: 2.6562130273077855e-10
step 766: Loss: 2.3579473928947436e-09
step 767: Loss: 2.6083363247053626e-10
step 768: Loss: 2.3208128752116863e-09
step 769: Loss: 2.578496305361e-10
step 770: Loss: 2.2838779756284566e-09
step 771: Loss: 2.5428675831662417e-10
step 772: Loss: 2.2455668435839016e-09
step 773: Loss: 2.503849350077303e-10
step 774: Loss: 2.2076371841706077e-09
step 775: Loss: 2.471831628270138e-10
step 776: Loss: 2.170512214405562e-09
step 777: Loss: 2.4373811302602633e-10
step 778: Loss: 2.1345250011961525e-09
step 779: Loss: 2.4058380287961256e-10
step 780: Loss: 2.098590634602715e-09
step 781: Loss: 2.3703872198410636e-10
step 782: Loss: 2.0650048337955695e-09
step 783: Loss: 2.3372911939212315e-10
step 784: Loss: 2.0304942172089113e-09
step 785: Loss: 2.3082939726304375e-10
step 786: Loss: 2.0007162593316252e-09
step 787: Loss: 2.2742069338832493e-10
step 788: Loss: 1.9673327411595665e-09
step 789: Loss: 2.2497677332200539e-10
step 790: Loss: 1.9367392134483907e-09
step 791: Loss: 2.2186898152032342e-10
step 792: Loss: 1.9098900239100658e-09
step 793: Loss: 2.1978809050526849e-10
step 794: Loss: 1.8807675417065184e-09
step 795: Loss: 2.1648684234154558e-10
step 796: Loss: 1.852985209715996e-09
step 797: Loss: 2.1362611679620613e-10
step 798: Loss: 1.8246666400045797e-09
step 799: Loss: 2.109122737570246e-10
step 800: Loss: 1.798781013029327e-09
step 801: Loss: 2.0829098168473337e-10
step 802: Loss: 1.7714745226271589e-09
step 803: Loss: 2.056037146092038e-10
step 804: Loss: 1.7464593105032122e-09
step 805: Loss: 2.0349133489361293e-10
step 806: Loss: 1.7232218985085979e-09
step 807: Loss: 2.0076050544215462e-10
step 808: Loss: 1.69969127661318e-09
step 809: Loss: 1.9940253614958436e-10
step 810: Loss: 1.6753238796241021e-09
step 811: Loss: 1.9698388753486284e-10
step 812: Loss: 1.6523672430324154e-09
step 813: Loss: 1.9384963079183137e-10
step 814: Loss: 1.6302768024445413e-09
step 815: Loss: 1.922346587468482e-10
step 816: Loss: 1.6067377428541363e-09
step 817: Loss: 1.8958976055749588e-10
```

```
step 818: Loss: 1.5840417866286316e-09
step 819: Loss: 1.8788744171605032e-10
step 820: Loss: 1.5624775917544298e-09
step 821: Loss: 1.8560018799629319e-10
step 822: Loss: 1.541340610700103 8e-09
step 823: Loss: 1.8320854555664567e-10
step 824: Loss: 1.5226305771776083e-09
step 825: Loss: 1.8075278773732606e-10
step 826: Loss: 1.5031976774437794e-09
step 827: Loss: 1.7858547973759187e-10
step 828: Loss: 1.4846918139355125e-09
step 829: Loss: 1.7684582964694329e-10
step 830: Loss: 1.4672990600317348e-09
step 831: Loss: 1.7509892147327122e-10
step 832: Loss: 1.4485255217522308e-09
step 833: Loss: 1.7245230243823073e-10
step 834: Loss: 1.4301522188731042e-09
step 835: Loss: 1.705292990150653e-10
step 836: Loss: 1.4122194524901488e-09
step 837: Loss: 1.683495565174553e-10
step 838: Loss: 1.3949769117616029e-09
step 839: Loss: 1.6671067304407927e-10
step 840: Loss: 1.377934322199792e-09
step 841: Loss: 1.6535649238758054e-10
step 842: Loss: 1.361845414216134e-09
step 843: Loss: 1.6350860942981882e-10
step 844: Loss: 1.3439531709735775e-09
step 845: Loss: 1.6124612756129864e-10
step 846: Loss: 1.3288741218531186e-09
step 847: Loss: 1.597880439074828e-10
step 848: Loss: 1.3128335085710319e-09
step 849: Loss: 1.5788002849514982e-10
step 850: Loss: 1.2955670980474565e-09
step 851: Loss: 1.5659806784640296e-10
step 852: Loss: 1.2813927696697647e-09
step 853: Loss: 1.5456227964172342e-10
step 854: Loss: 1.266821203493862e-09
step 855: Loss: 1.5318243895556805e-10
step 856: Loss: 1.2525160908438693e-09
step 857: Loss: 1.5127209207488335e-10
step 858: Loss: 1.2380616531970645e-09
step 859: Loss: 1.5010698239947828e-10
step 860: Loss: 1.2237423296923566e-09
step 861: Loss: 1.4889119104299908e-10
step 862: Loss: 1.208742772540461e-09
step 863: Loss: 1.4729602260121766e-10
step 864: Loss: 1.1956509116117786e-09
step 865: Loss: 1.4627594968619206e-10
step 866: Loss: 1.1833711788256096e-09
step 867: Loss: 1.4460918573711012e-10
step 868: Loss: 1.1708878311367243e-09
step 869: Loss: 1.4321072105971666e-10
step 870: Loss: 1.1570890912082632e-09
```

```
step 871: Loss: 1.4143863857896122e-10
step 872: Loss: 1.144060624014287e-09
step 873: Loss: 1.4006437676350458e-10
step 874: Loss: 1.131678861732155e-09
step 875: Loss: 1.3860385061903457e-10
step 876: Loss: 1.1195351312665025e-09
step 877: Loss: 1.381077058271174e-10
step 878: Loss: 1.1067052829716317e-09
step 879: Loss: 1.3579443125522062e-10
step 880: Loss: 1.0951392015456918e-09
step 881: Loss: 1.3427828293721689e-10
step 882: Loss: 1.0838687725112095e-09
step 883: Loss: 1.332560867206567e-10
step 884: Loss: 1.0708081088495192e-09
step 885: Loss: 1.3166320811386356e-10
step 886: Loss: 1.0601662880915796e-09
step 887: Loss: 1.3049752944915838e-10
step 888: Loss: 1.0489218382758736e-09
step 889: Loss: 1.2899152579404216e-10
step 890: Loss: 1.0384839654875577e-09
step 891: Loss: 1.2792759906954387e-10
step 892: Loss: 1.0277464435048955e-09
step 893: Loss: 1.2654555181512706e-10
step 894: Loss: 1.0183758281101518e-09
step 895: Loss: 1.2546420846692996e-10
step 896: Loss: 1.00858466023368e-09
step 897: Loss: 1.2468992505176857e-10
step 898: Loss: 9.97817384273958e-10
step 899: Loss: 1.2358421230818095e-10
step 900: Loss: 9.87902759597148e-10
step 901: Loss: 1.2265369275787918e-10
step 902: Loss: 9.774522302663513e-10
step 903: Loss: 1.2144107941480797e-10
step 904: Loss: 9.688080337966198e-10
step 905: Loss: 1.2083745115631928e-10
step 906: Loss: 9.583053239836659e-10
step 907: Loss: 1.1957614065583044e-10
step 908: Loss: 9.502268971672834e-10
step 909: Loss: 1.1928096010915823e-10
step 910: Loss: 9.411618151489165e-10
step 911: Loss: 1.1790395049171565e-10
step 912: Loss: 9.309939486001895e-10
step 913: Loss: 1.1743810091058293e-10
step 914: Loss: 9.214697338499889e-10
step 915: Loss: 1.1575912450823012e-10
step 916: Loss: 9.128005573622033e-10
step 917: Loss: 1.1513515141281516e-10
step 918: Loss: 9.040595494447246e-10
step 919: Loss: 1.1353117751466968e-10
step 920: Loss: 8.957798947051288e-10
step 921: Loss: 1.1309374270407346e-10
step 922: Loss: 8.875238877159575e-10
step 923: Loss: 1.1224709356438822e-10
```

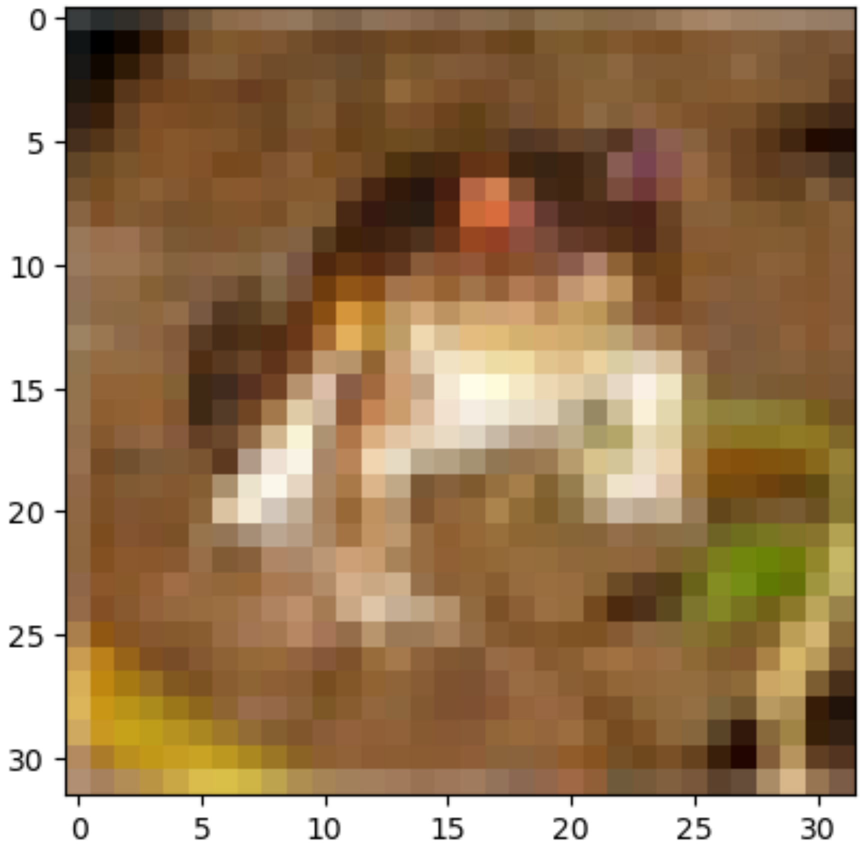
```
step 924: Loss: 8.785547844780695e-10
step 925: Loss: 1.1089622969917556e-10
step 926: Loss: 8.705833276501096e-10
step 927: Loss: 1.1082874895596007e-10
step 928: Loss: 8.608989632286068e-10
step 929: Loss: 1.0949127715598195e-10
step 930: Loss: 8.534465911758105e-10
step 931: Loss: 1.0860879556817693e-10
step 932: Loss: 8.459807299132649e-10
step 933: Loss: 1.0766845748300113e-10
step 934: Loss: 8.377037952200794e-10
step 935: Loss: 1.0687694479427634e-10
step 936: Loss: 8.299750331453026e-10
step 937: Loss: 1.0530438326883385e-10
step 938: Loss: 8.219708247381163e-10
step 939: Loss: 1.0504135755651234e-10
step 940: Loss: 8.149662611423025e-10
step 941: Loss: 1.0332370375953914e-10
step 942: Loss: 8.071616153237926e-10
step 943: Loss: 1.0308844750062107e-10
step 944: Loss: 7.997966733341855e-10
step 945: Loss: 1.0178333176291687e-10
step 946: Loss: 7.925577416578733e-10
step 947: Loss: 1.0072701006613727e-10
step 948: Loss: 7.853360184384428e-10
step 949: Loss: 1.0033505970508116e-10
step 950: Loss: 7.786924993702371e-10
step 951: Loss: 9.909745246616808e-11
step 952: Loss: 7.713554239785481e-10
step 953: Loss: 9.850506521580371e-11
step 954: Loss: 7.645392652300131e-10
step 955: Loss: 9.799422384659806e-11
step 956: Loss: 7.569119220285359e-10
step 957: Loss: 9.727799121783676e-11
step 958: Loss: 7.506116839195442e-10
step 959: Loss: 9.636357684028596e-11
step 960: Loss: 7.441051108614261e-10
step 961: Loss: 9.600853445590474e-11
step 962: Loss: 7.376331212505249e-10
step 963: Loss: 9.480682211515656e-11
step 964: Loss: 7.306243943183688e-10
step 965: Loss: 9.42563527228657e-11
step 966: Loss: 7.241833244187035e-10
step 967: Loss: 9.335840434054887e-11
step 968: Loss: 7.180255279237713e-10
step 969: Loss: 9.303233183821646e-11
step 970: Loss: 7.114914768457936e-10
step 971: Loss: 9.24463422480315e-11
step 972: Loss: 7.054787309890287e-10
step 973: Loss: 9.203812018077073e-11
step 974: Loss: 6.988414846809121e-10
step 975: Loss: 9.142375051451879e-11
step 976: Loss: 6.949807951350806e-10
```

```
step 977: Loss: 9.070013490264373e-11
step 978: Loss: 6.883335568197424e-10
step 979: Loss: 8.991090511001332e-11
step 980: Loss: 6.830466747764774e-10
step 981: Loss: 8.92292698062569e-11
step 982: Loss: 6.773600569331961e-10
step 983: Loss: 8.858722783111617e-11
step 984: Loss: 6.717100209385762e-10
step 985: Loss: 8.826587377663841e-11
step 986: Loss: 6.660704765515391e-10
step 987: Loss: 8.751873531442911e-11
step 988: Loss: 6.605767599587864e-10
step 989: Loss: 8.676635104842845e-11
step 990: Loss: 6.551614806227235e-10
step 991: Loss: 8.587763139500382e-11
step 992: Loss: 6.500540661313892e-10
step 993: Loss: 8.575751220263328e-11
step 994: Loss: 6.447729572478522e-10
step 995: Loss: 8.485066815833164e-11
step 996: Loss: 6.389547779761529e-10
step 997: Loss: 8.451319505331512e-11
step 998: Loss: 6.327757207102991e-10
step 999: Loss: 8.342503771130438e-11
minimal loss achieved: 8.342503771130438e-11
```

In [19]:

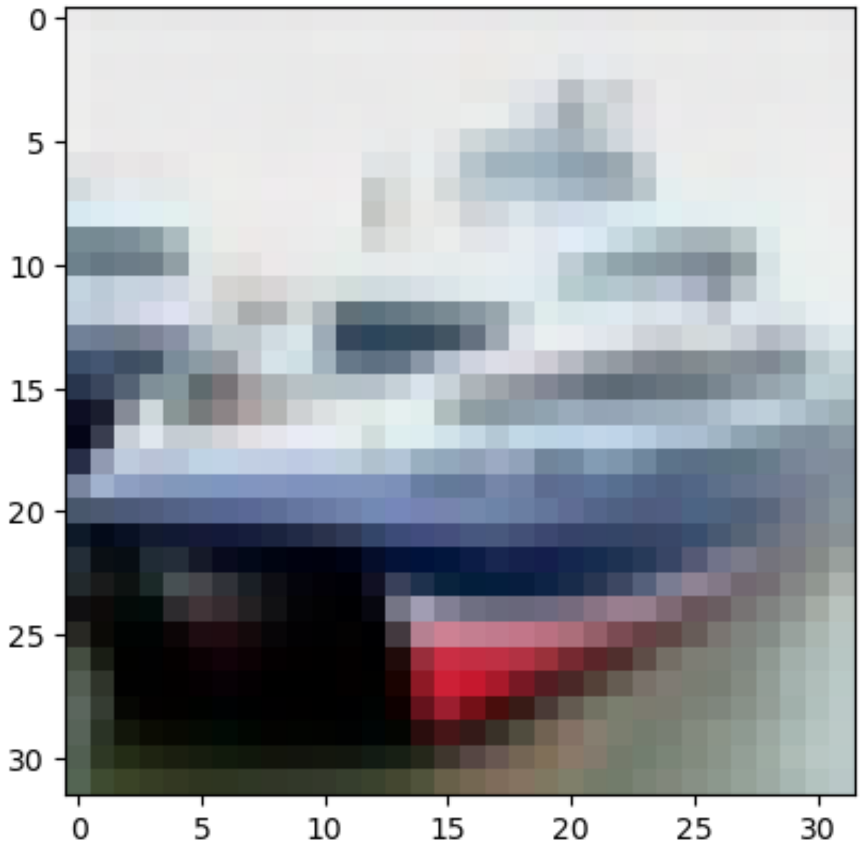
```
pred = f(z1)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

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

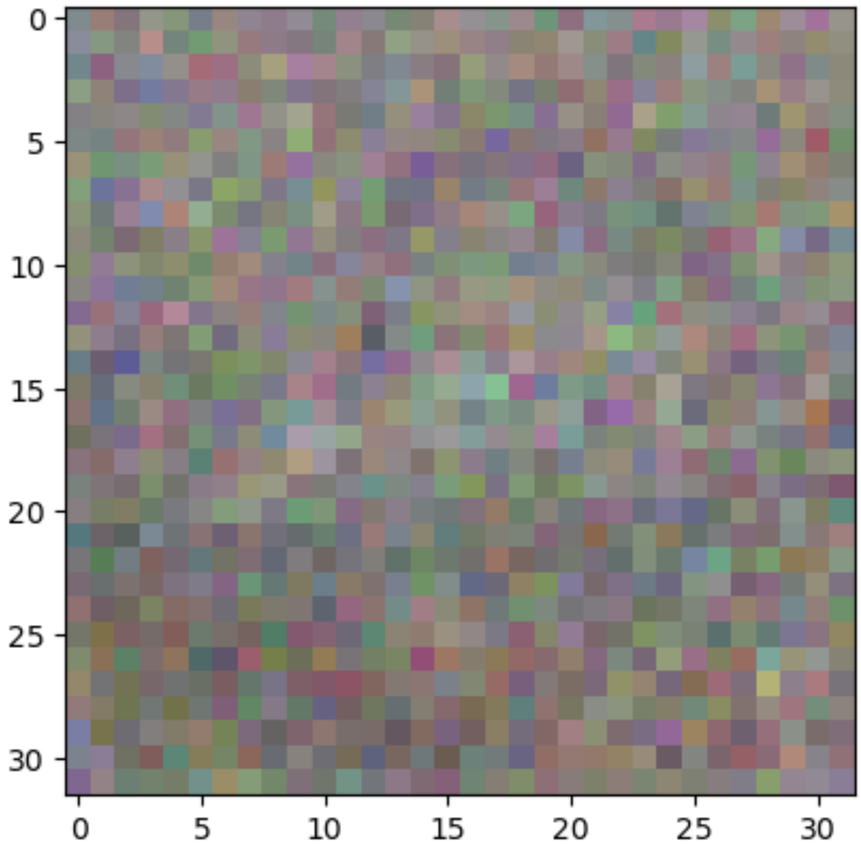


In [20]:
```
pred = f(z2)
pred = pred.reshape_as(z2)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

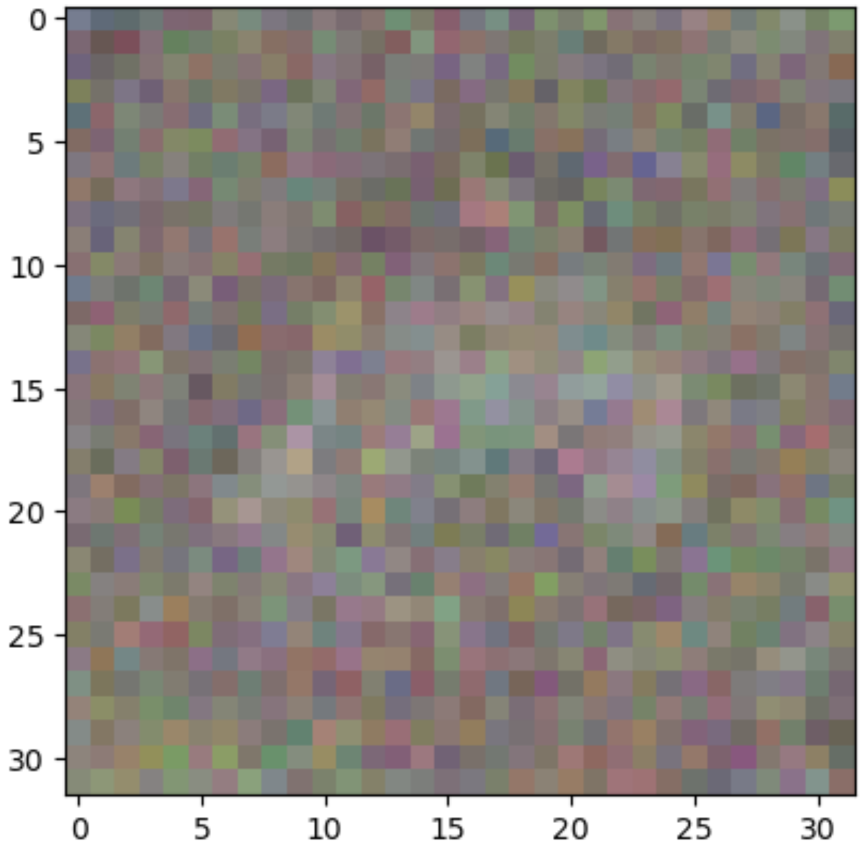
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



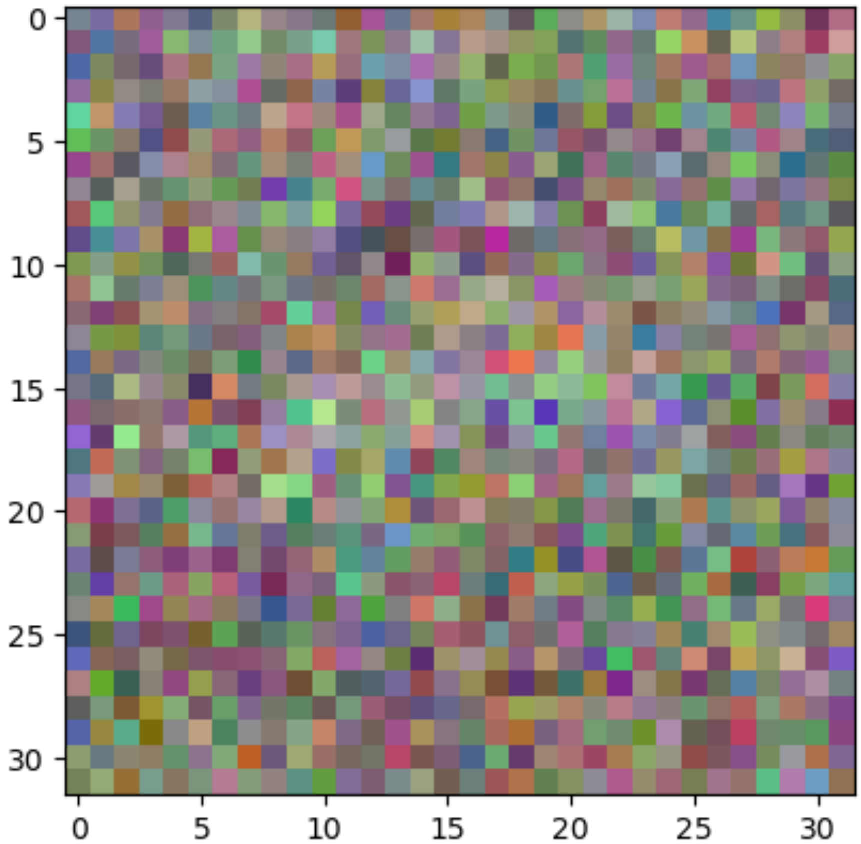
In [21]:
```
pred = f(random_train)
pred = pred.reshape_as(random_train)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



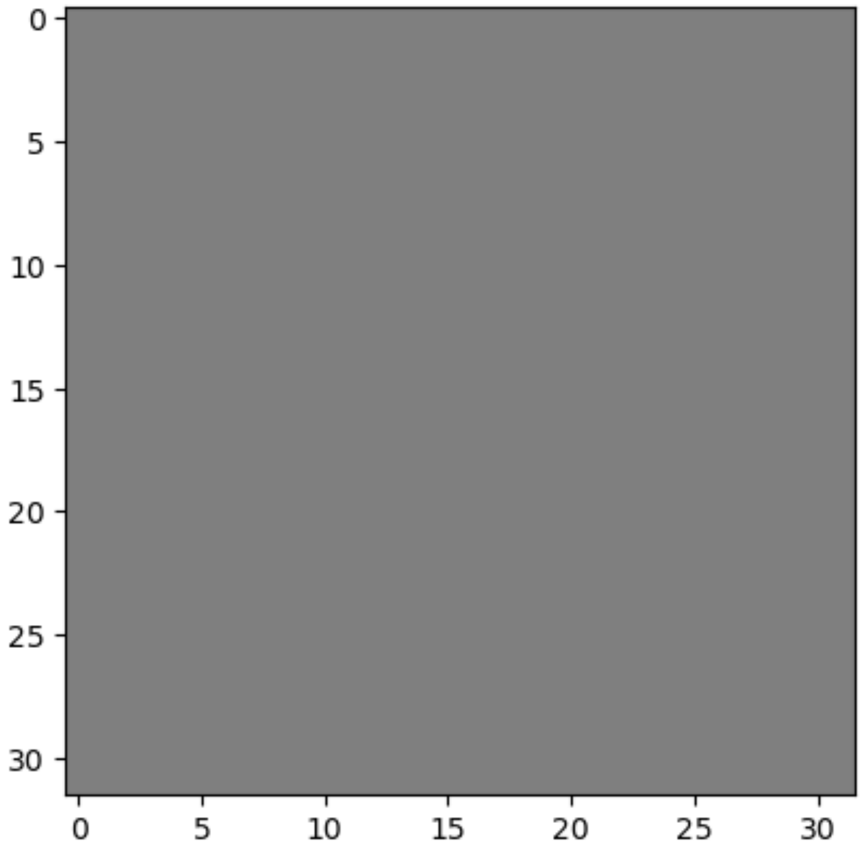
In [22]:
```
pred = f(random_test)
pred = pred.reshape_as(random_train)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



In [23]:
```
pred = f(noise)
pred = pred.reshape_as(noise)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



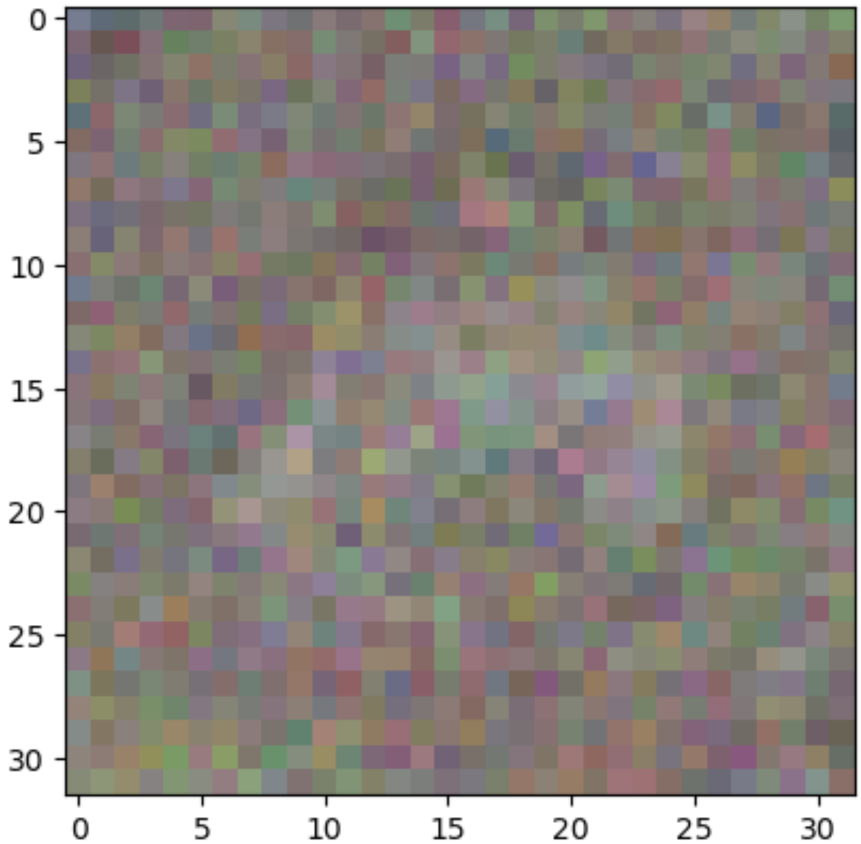
In [24]:
```
pred = f(zeros)
pred = pred.reshape_as(zeros)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



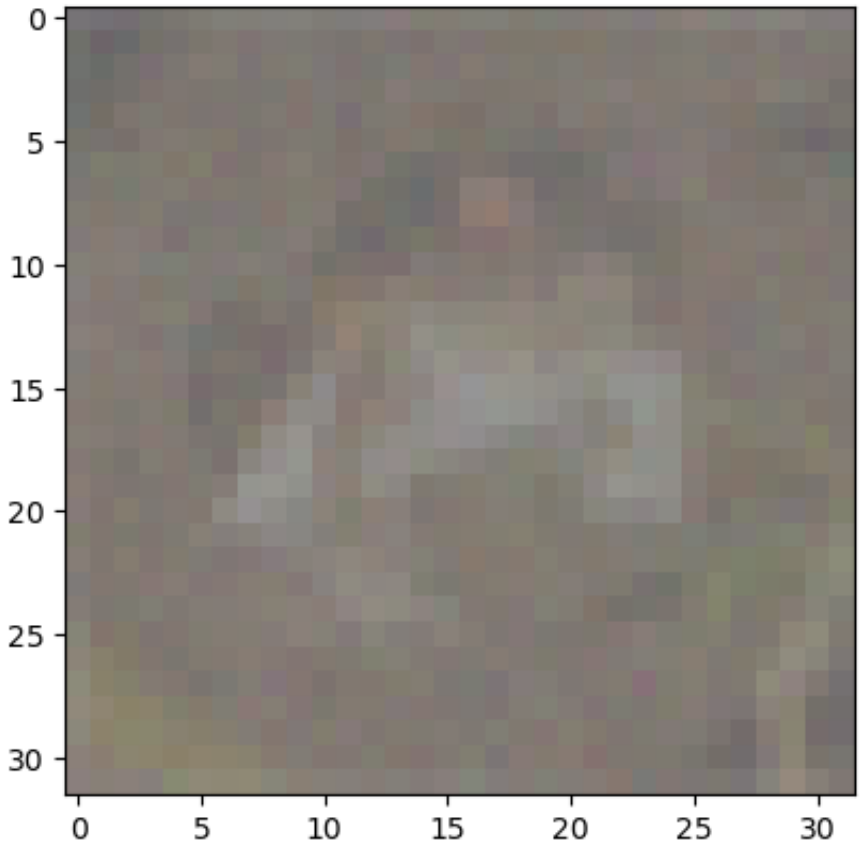
(d) $f(x), f^2(x), f^{20}(x)$

In [25]:
```
def fn(x ,n, f):
    for i in range(n):
        x = f(x)
    return x
```

In [26]:
```
pred = fn(random_test, 1, f)
pred = pred.reshape_as(zeros)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

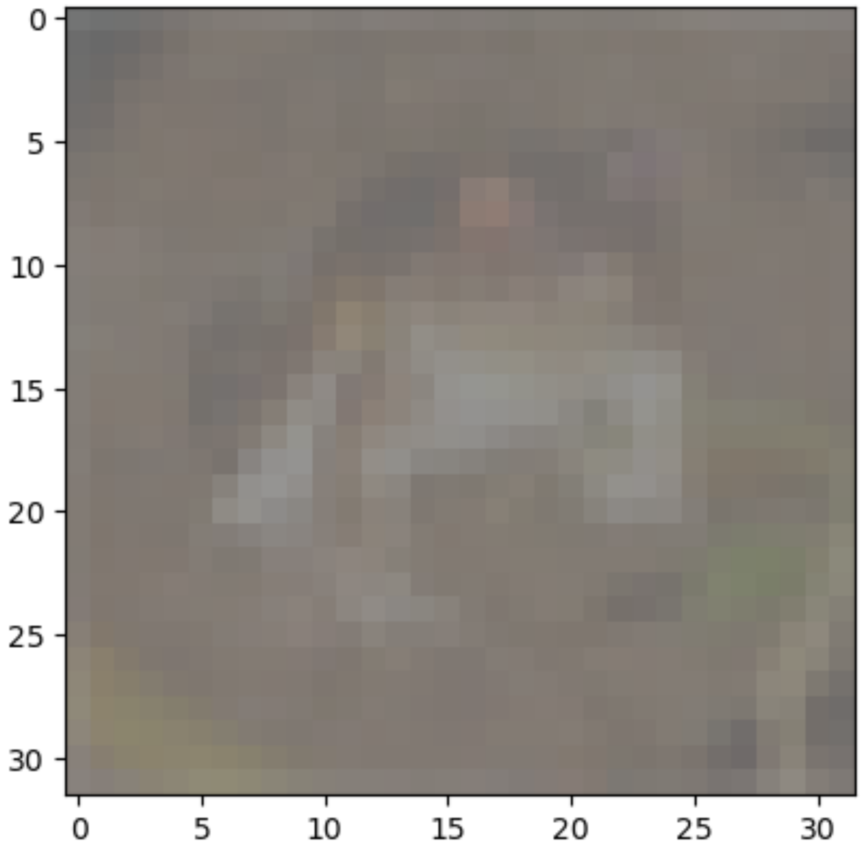


In [27]:
```
pred = fn(random_test, 2, f)
pred = pred.reshape_as(zeros)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



In [28]:

```
pred = fn(random_test, 20, f)
pred = pred.reshape_as(zeros)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



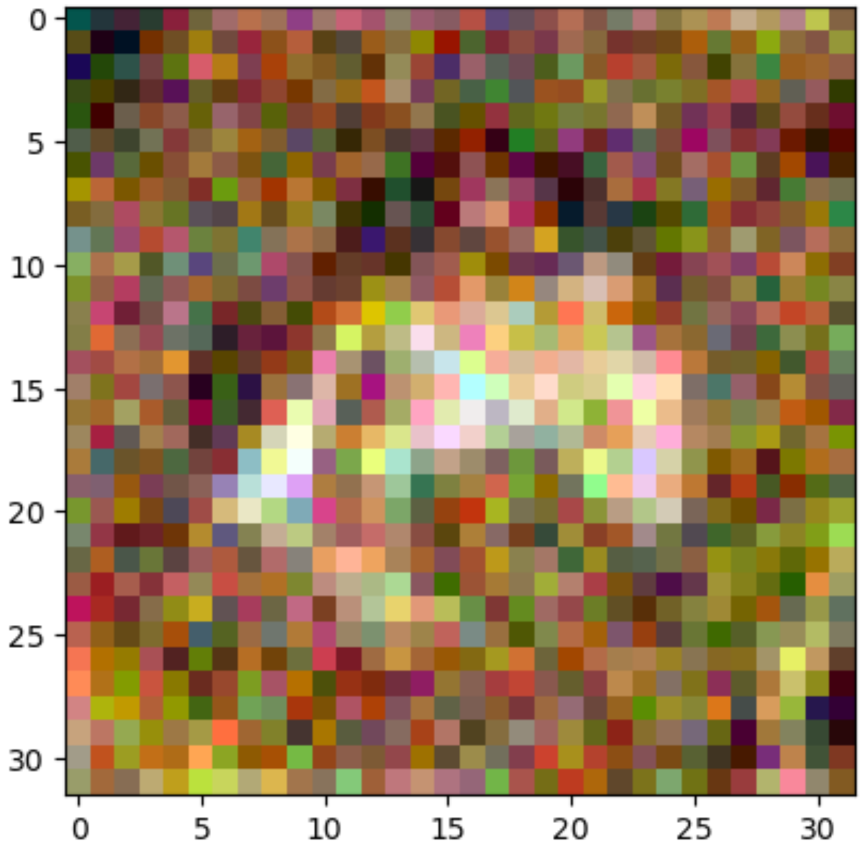
They become more close to the z_1 as n increases (but why not close to z_2?). Honestly, I don't understand why this happens theoretically but want to understand it.

(e)

In [29]:
```
def add_noise(x):
    return x + torch.normal(0, 1 , size=(1, 3, 32, 32), requires_grad=False)
```

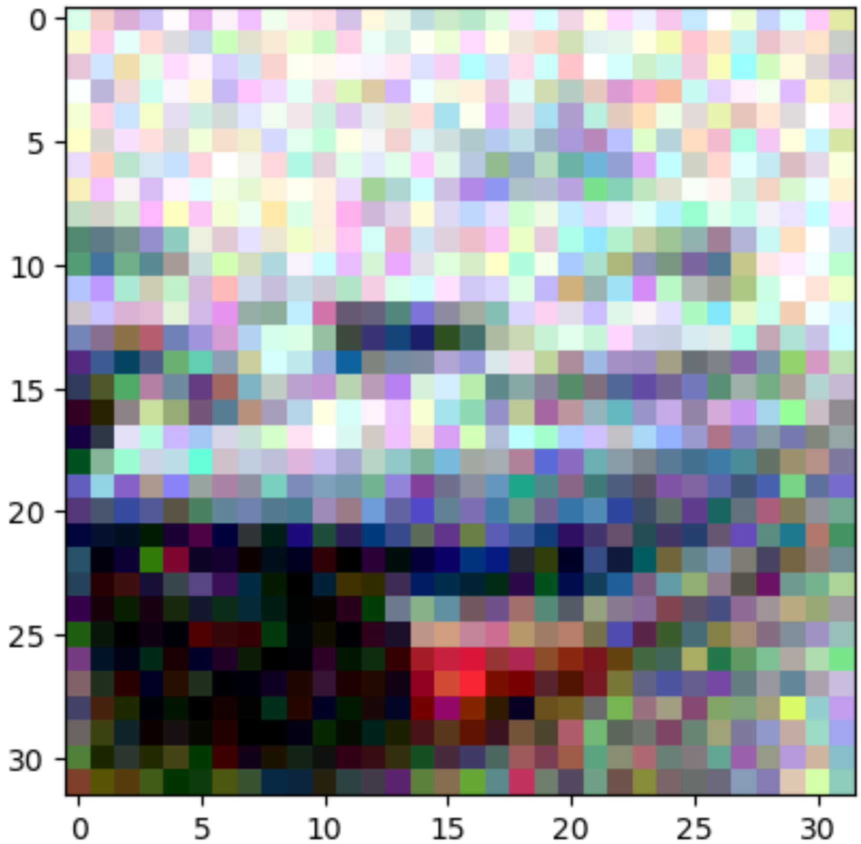
In [30]:
```
pred = fn(add_noise(z1), 1, f)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

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



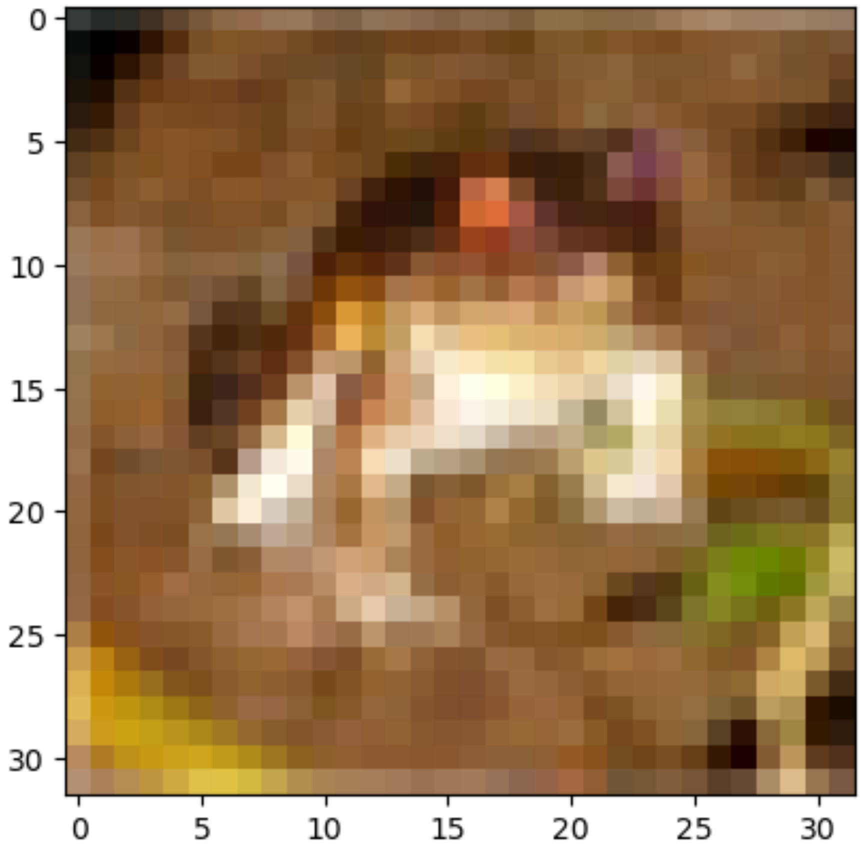
In [31]:
```
pred = fn(add_noise(z2), 1, f)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

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

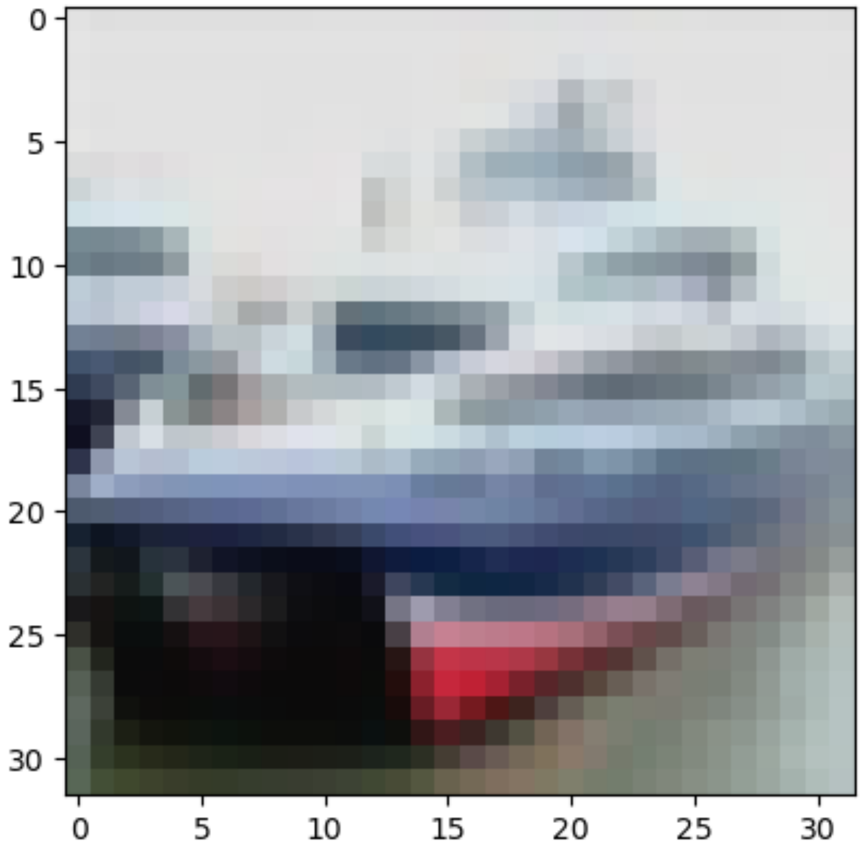


In [32]:
```
pred = fn(add_noise(z1), 10, f)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```

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



In [33]:
```
pred = fn(add_noise(z2), 10, f)
pred = pred.reshape_as(z1)
pred = pred.detach()
imshow(torchvision.utils.make_grid(pred))
```



Noise makes the reconstruction by autoencoder more clear. (but why?)

Problem 5

Wednesday, January 25, 2023 3:35 PM

$$\psi(\varepsilon) = \mathbb{E}[\phi(u)\phi(v)]$$

$$= \mathbb{E}[u \cdot v]$$

$$= \mathbb{E}[u \cdot v] - \mathbb{E}[u] \cdot \mathbb{E}[v] \quad (\because \mathbb{E}[u] \cdot \mathbb{E}[v] = 0)$$

$$= \mathrm{cov}(u, v)$$

$$= \varepsilon \quad \square \text{ QED}$$