model_132_beta=1_(d_k)^0_5_d_k=20_without_using_the_alter_algo

September 26, 2023

[4]:
```python
# Author: William Chuang
# Last modified: Sep 26, 2023
# This notebook is built on the code written by Dr. Phillip Lippe.
#
## Standard libraries
import os
import numpy as np
import random
import math
import json
from functools import partial
import statistics as stat

## PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as data
import torch.optim as optim


# PyTorch Lightning
try:
    import pytorch_lightning as pl
except ModuleNotFoundError: # Google Colab does not have PyTorch Lightning␣
↪installed by default. Hence, we do it here if necessary
    !pip install --quiet pytorch-lightning>=1.4
    import pytorch_lightning as pl
from pytorch_lightning.callbacks import LearningRateMonitor, ModelCheckpoint

# Path to the folder where the datasets are/should be downloaded (e.g. CIFAR10)
DATASET_PATH = "./data"
# Path to the folder where the pretrained models are saved
CHECKPOINT_PATH = "./saved_models/tutorial6"
```

```
# Setting the seed
pl.seed_everything(42)

# Ensure that all operations are deterministic on GPU (if used) for␣
↪reproducibility
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.
↪device("cpu")
print("Device:", device)


def scaled_dot_product(q, k, v, mask=None):
    d_k = q.size()[-1]
    PATH = "./tmp.pth"

    #torch.save(reverse_model.state_dict(), PATH)
    #w=torch.load(PATH)
    #d=sigma=torch.std((w["transformer.layers.0.self_attn.qkv_proj.weight"])).
↪item()
    #print(d_k)
    attn_logits = torch.matmul(q, k.transpose(-2, -1))
    attn_logits = attn_logits / (math.sqrt(d_k)) #math.sqrt(0.005*d_k) #0.005␣
↪#10 #3 #1.414 #(0.00001*d_k) #(d_k)**(1/100) #math.sqrt(d_k*2)
    #print(attn_logits)
    if mask is not None:
        attn_logits = attn_logits.masked_fill(mask == 0, -9e15)
    attention = F.softmax(attn_logits, dim=-1)
    values = torch.matmul(attention, v)
    return values, attention
class MultiheadAttention(nn.Module):

    def __init__(self, input_dim, embed_dim, num_heads):
        super().__init__()
        assert embed_dim % num_heads == 0, "Embedding dimension must be 0␣
↪modulo number of heads."

        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.head_dim = embed_dim // num_heads

        # Stack all weight matrices 1...h together for efficiency
        # Note that in many implementations you see "bias=False" which is␣
↪optional
        self.qkv_proj = nn.Linear(input_dim, 3*embed_dim)
```

```
        self.o_proj = nn.Linear(embed_dim, embed_dim)

        self._reset_parameters()

    def _reset_parameters(self):
        # Original Transformer initialization, see PyTorch documentation
        nn.init.xavier_uniform_(self.qkv_proj.weight)
        self.qkv_proj.bias.data.fill_(0)
        nn.init.xavier_uniform_(self.o_proj.weight)
        self.o_proj.bias.data.fill_(0)

    def forward(self, x, mask=None, return_attention=False):
        batch_size, seq_length, _ = x.size()
        if mask is not None:
            mask = expand_mask(mask)
        qkv = self.qkv_proj(x)

        # Separate Q, K, V from linear output
        qkv = qkv.reshape(batch_size, seq_length, self.num_heads, 3*self.
↪head_dim)
        qkv = qkv.permute(0, 2, 1, 3) # [Batch, Head, SeqLen, Dims]
        q, k, v = qkv.chunk(3, dim=-1)


        # Determine value outputs
        values, attention = scaled_dot_product(q, k, v, mask=mask)
        values = values.permute(0, 2, 1, 3) # [Batch, SeqLen, Head, Dims]
        values = values.reshape(batch_size, seq_length, self.embed_dim)
        o = self.o_proj(values)

        if return_attention:
            return o, attention
        else:
            return o
class EncoderBlock(nn.Module):

    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        """
        Inputs:
            input_dim - Dimensionality of the input
            num_heads - Number of heads to use in the attention block
            dim_feedforward - Dimensionality of the hidden layer in the MLP
            dropout - Dropout probability to use in the dropout layers
        """
        super().__init__()

        # Attention layer
```

```
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Two-layer MLP
        self.linear_net = nn.Sequential(
            nn.Linear(input_dim, dim_feedforward),
            nn.Dropout(dropout),
            nn.ReLU(inplace=True),
            nn.Linear(dim_feedforward, input_dim)
        )

        # Layers to apply in between the main layers
        self.norm1 = nn.LayerNorm(input_dim)
        self.norm2 = nn.LayerNorm(input_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        # Attention part
        attn_out = self.self_attn(x, mask=mask)
        x = x + self.dropout(attn_out)
        x = self.norm1(x)

        # MLP part
        linear_out = self.linear_net(x)
        x = x + self.dropout(linear_out)
        x = self.norm2(x)

        return x
class TransformerEncoder(nn.Module):

    def __init__(self, num_layers, **block_args):
        super().__init__()
        self.layers = nn.ModuleList([EncoderBlock(**block_args) for _ in
 ↪range(num_layers)])

    def forward(self, x, mask=None):
        for l in self.layers:
            x = l(x, mask=mask)
        return x

    def get_attention_maps(self, x, mask=None):
        attention_maps = []
        for l in self.layers:
            _, attn_map = l.self_attn(x, mask=mask, return_attention=True)
            attention_maps.append(attn_map)
            x = l(x)
        return attention_maps
class PositionalEncoding(nn.Module):
```

```
    def __init__(self, d_model, max_len=5000):
        """
        Inputs
            d_model - Hidden dimensionality of the input.
            max_len - Maximum length of a sequence to expect.
        """
        super().__init__()

        # Create matrix of [SeqLen, HiddenDim] representing the positional␣
↪encoding for max_len inputs
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.
↪log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)

        # register_buffer => Tensor which is not a parameter, but should be␣
↪part of the modules state.
        # Used for tensors that need to be on the same device as the module.
        # persistent=False tells PyTorch to not add the buffer to the state␣
↪dict (e.g. when we save the model)
        self.register_buffer('pe', pe, persistent=False)

    def forward(self, x):
        x = x + self.pe[:, :x.size(1)]
        return x
class TransformerPredictor(pl.LightningModule):

    def __init__(self, input_dim, model_dim, num_classes, num_heads,␣
↪num_layers, lr, warmup, max_iters, dropout=0.0, input_dropout=0.0):
        """
        Inputs:
            input_dim - Hidden dimensionality of the input
            model_dim - Hidden dimensionality to use inside the Transformer
            num_classes - Number of classes to predict per sequence element
            num_heads - Number of heads to use in the Multi-Head Attention␣
↪blocks
            num_layers - Number of encoder blocks to use.
            lr - Learning rate in the optimizer
            warmup - Number of warmup steps. Usually between 50 and 500
            max_iters - Number of maximum iterations the model is trained for.␣
↪This is needed for the CosineWarmup scheduler
            dropout - Dropout to apply inside the model
```

```
            input_dropout - Dropout to apply on the input features
        """
        super().__init__()
        self.save_hyperparameters()
        self._create_model()

    def _create_model(self):
        # Input dim -> Model dim
        self.input_net = nn.Sequential(
            nn.Dropout(self.hparams.input_dropout),
            nn.Linear(self.hparams.input_dim, self.hparams.model_dim)
        )
        # Positional encoding for sequences
        self.positional_encoding = PositionalEncoding(d_model=self.hparams.
↪model_dim)
        # Transformer
        self.transformer = TransformerEncoder(num_layers=self.hparams.
↪num_layers,
                                              input_dim=self.hparams.model_dim,
                                              dim_feedforward=2*self.hparams.
↪model_dim,
                                              num_heads=self.hparams.num_heads,
                                              dropout=self.hparams.dropout)
        # Output classifier per sequence lement
        self.output_net = nn.Sequential(
            nn.Linear(self.hparams.model_dim, self.hparams.model_dim),
            nn.LayerNorm(self.hparams.model_dim),
            nn.ReLU(inplace=True),
            nn.Dropout(self.hparams.dropout),
            nn.Linear(self.hparams.model_dim, self.hparams.num_classes)
        )

    def forward(self, x, mask=None, add_positional_encoding=True):
        """
        Inputs:
            x - Input features of shape [Batch, SeqLen, input_dim]
            mask - Mask to apply on the attention outputs (optional)
            add_positional_encoding - If True, we add the positional encoding␣
↪to the input.
                                      Might not be desired for some tasks.
        """
        x = self.input_net(x)
        if add_positional_encoding:
            x = self.positional_encoding(x)
        x = self.transformer(x, mask=mask)
        x = self.output_net(x)
        return x
```

```
    @torch.no_grad()
    def get_attention_maps(self, x, mask=None, add_positional_encoding=True):
        """
        Function for extracting the attention matrices of the whole Transformer␣
↪for a single batch.
        Input arguments same as the forward pass.
        """
        x = self.input_net(x)
        if add_positional_encoding:
            x = self.positional_encoding(x)
        attention_maps = self.transformer.get_attention_maps(x, mask=mask)
        return attention_maps

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(), lr=self.hparams.lr)

        # Apply lr scheduler per step
        lr_scheduler = CosineWarmupScheduler(optimizer,
                                             warmup=self.hparams.warmup,
                                             max_iters=self.hparams.max_iters)
        return [optimizer], [{'scheduler': lr_scheduler, 'interval': 'step'}]

    def training_step(self, batch, batch_idx):
        raise NotImplementedError

    def validation_step(self, batch, batch_idx):
        raise NotImplementedError

    def test_step(self, batch, batch_idx):
        raise NotImplementedError
class ReverseDataset(data.Dataset):

    def __init__(self, num_categories, seq_len, size):
        super().__init__()
        self.num_categories = num_categories

        self.seq_len = seq_len
        self.size = size

        self.data = torch.randint(self.num_categories, size=(self.size, self.
↪seq_len))
        # self.data = torch.abs(torch.normal(0, 1, size=(self.size, self.
↪seq_len)).long())
        # torch.randint(low=0, high, size, \*, generator=None, out=None,␣
↪dtype=None,
        # layout=torch.strided, device=None, requires_grad=False) → Tensor
```

```
        print(self.num_categories)
        print(self.seq_len)
        print(self.size)
        print(self.data)

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        inp_data = self.data[idx]
        labels = torch.flip(inp_data, dims=(0,))
        return inp_data, labels
#''' Examples of torch.randint
#>>> torch.randint(3, 5, (3,))
#tensor([4, 3, 4])


#>>> torch.randint(10, (2, 2))
#tensor([[0, 2],
#         [5, 5]])


#>>> torch.randint(3, 10, (2, 2))
#tensor([[4, 5],
#         [6, 7]])'''

#'''>>> torch.normal(mean=0.5, std=torch.arange(1., 6.))
#tensor([-1.2793, -1.0732, -2.0687,  5.1177, -1.2303])'''

class ReversePredictor(TransformerPredictor):

    def _calculate_loss(self, batch, mode="train"):
        # Fetch data and transform categories to one-hot vectors
        inp_data, labels = batch
        inp_data = F.one_hot(inp_data, num_classes=self.hparams.num_classes).
 ↪float()

        # Perform prediction and calculate loss and accuracy
        preds = self.forward(inp_data, add_positional_encoding=True)
        loss = F.cross_entropy(preds.view(-1,preds.size(-1)), labels.view(-1))
        acc = (preds.argmax(dim=-1) == labels).float().mean()

        # Logging
        self.log(f"{mode}_loss", loss)
        self.log(f"{mode}_acc", acc)
        return loss, acc
```

```python
    def training_step(self, batch, batch_idx):
        loss, _ = self._calculate_loss(batch, mode="train")
        return loss

    def validation_step(self, batch, batch_idx):
        _ = self._calculate_loss(batch, mode="val")

    def test_step(self, batch, batch_idx):
        _ = self._calculate_loss(batch, mode="test")
def train_reverse(**kwargs):

    # Create a PyTorch Lightning trainer with the generation callback
    root_dir = os.path.join(CHECKPOINT_PATH, "ReverseTask")
    os.makedirs(root_dir, exist_ok=True)
    trainer = pl.Trainer(default_root_dir=root_dir,
                         callbacks=[ModelCheckpoint(save_weights_only=True,␣
 ↪mode="max", monitor="val_acc")],
                         accelerator="gpu" if str(device).startswith("cuda")␣
 ↪else "cpu",
                         devices=1,
                         max_epochs=10,
                         gradient_clip_val=5)
    trainer.logger._default_hp_metric = None # Optional logging argument that␣
 ↪we don't need
    trainer.callbacks
    # Check whether pretrained model exists. If yes, load it and skip training
    pretrained_filename = os.path.join(CHECKPOINT_PATH, "ReverseTask.ckpt")
    if os.path.isfile(pretrained_filename):
        print("Found pretrained model, loading...")
        model = ReversePredictor.load_from_checkpoint(pretrained_filename)
    else:
        print("Found pretrained model does not exist, generating...")
        model = ReversePredictor(max_iters=trainer.
 ↪max_epochs*len(train_loader), **kwargs)
        trainer.fit(model, train_loader, val_loader)

    # Test best model on validation and test set
    val_result = trainer.test(model, val_loader, verbose=False)
    test_result = trainer.test(model, test_loader, verbose=False)
    result = {"test_acc": test_result[0]["test_acc"], "val_acc":␣
 ↪val_result[0]["test_acc"]}

    model = model.to(device)
    return model, result
class CosineWarmupScheduler(optim.lr_scheduler._LRScheduler):

    def __init__(self, optimizer, warmup, max_iters):
```

```python
        self.warmup = warmup
        self.max_num_iters = max_iters
        super().__init__(optimizer)

    def get_lr(self):
        lr_factor = self.get_lr_factor(epoch=self.last_epoch)
        return [base_lr * lr_factor for base_lr in self.base_lrs]

    def get_lr_factor(self, epoch):
        lr_factor = 0.5 * (1 + np.cos(np.pi * epoch / self.max_num_iters))
        if epoch <= self.warmup:
            lr_factor *= epoch * 1.0 / self.warmup
        return lr_factor

dataset = partial(ReverseDataset, 100, 20)
train_loader = data.DataLoader(dataset(15000), batch_size=128, shuffle=True,␣
 ↪drop_last=True, pin_memory=True)
val_loader   = data.DataLoader(dataset(1000), batch_size=128)
test_loader  = data.DataLoader(dataset(100000), batch_size=128)
inp_data, labels = train_loader.dataset[0]
print("Input data:", inp_data)
print("Labels:    ", labels)
```

```
INFO:lightning_fabric.utilities.seed:Global seed set to 42
```

```
Device: cuda:0
100
20
15000
tensor([[42, 67, 76,  …, 95, 67,  6],
        [49, 76, 73,  …, 76, 32, 10],
        [86, 22, 77,  …, 84, 78,  8],
        …,
        [ 1, 31, 80,  …, 38, 66, 80],
        [37, 57, 93,  …, 31,  8, 65],
        [23, 88, 33,  …, 57, 23, 51]])
100
20
1000
tensor([[ 6, 25, 53,  …, 40, 80, 91],
        [30, 12, 95,  …, 42, 22, 95],
        [50, 62, 71,  …, 39, 51, 82],
        …,
        [45, 54,  2,  …, 76, 24, 92],
        [ 1, 50, 81,  …,  3, 23, 81],
        [93, 89, 53,  …, 37, 78, 83]])
100
20
```

```
100000
tensor([[83, 59, 87,  …, 50, 18, 19],
        [90, 85, 37,  …, 31, 49, 62],
        [22,  2, 57,  …, 91, 44, 21],
        …,
        [14,  9, 53,  …, 34, 61, 49],
        [20, 83, 70,  …, 61, 43, 64],
        [59, 97, 69,  …, 28, 75, 83]])
Input data: tensor([42, 67, 76, 14, 26, 35, 20, 24, 50, 13, 78, 14, 10, 54, 31,
72, 15, 95,
        67,  6])
Labels:     tensor([ 6, 67, 95, 15, 72, 31, 54, 10, 14, 78, 13, 50, 24, 20, 35,
26, 14, 76,
        67, 42])
```

[5]:
```python
reverse_model, reverse_result = train_reverse(input_dim=train_loader.dataset.
 ↪num_categories,
                                              model_dim=20,
                                              num_heads=1,
                                              num_classes=train_loader.dataset.
 ↪num_categories,
                                              num_layers=1,
                                              dropout=0.0,
                                              lr=5e-4,
                                              warmup=50)
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name                | Type               | Params
-----------------------------------------------------------
0 | input_net           | Sequential         | 2.0 K
1 | positional_encoding | PositionalEncoding | 0
2 | transformer         | TransformerEncoder | 3.4 K
3 | output_net          | Sequential         | 2.6 K
-----------------------------------------------------------
8.0 K     Trainable params
0         Non-trainable params
8.0 K     Total params
0.032     Total estimated model params size (MB)
```

```
Found pretrained model does not exist, generating…
```

```
Sanity Checking: 0it [00:00, ?it/s]

Training: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

Validation: 0it [00:00, ?it/s]

INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=10` reached.
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]

Testing: 0it [00:00, ?it/s]

INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]

Testing: 0it [00:00, ?it/s]
```

[6]:

```
# @title the scaling factor, beta, is 1/(d_k)^0.5, where d_k = 20, i.e. without␣
↪using the alternative algorithm to obtain the scaling factor first
# model_132_beta=1/(d_k)^0.5_d_k=20_without_using_the_alter_algo.ipynb
print(f"Val accuracy:  {(100.0 * reverse_result['val_acc']):4.2f}%")
print(f"Test accuracy: {(100.0 * reverse_result['test_acc']):4.2f}%")
```

```
Val accuracy:  1.52%
Test accuracy: 1.50%
```

[7]:

```
PATH = "./new.pth"
torch.save(reverse_model.state_dict(), PATH)
PATH = "./new.pth"
w=torch.load(PATH)
```

[8]:

```
#for odict in w:
#    print(odict)
for odict in w:
    print(str(odict)+": "+str(len(w[str(odict)])))
    #print(str( (w[str(odict)][5])))
```

```
input_net.1.weight: 20
input_net.1.bias: 20
transformer.layers.0.self_attn.qkv_proj.weight: 60
transformer.layers.0.self_attn.qkv_proj.bias: 60
transformer.layers.0.self_attn.o_proj.weight: 20
transformer.layers.0.self_attn.o_proj.bias: 20
transformer.layers.0.linear_net.0.weight: 40
transformer.layers.0.linear_net.0.bias: 40
transformer.layers.0.linear_net.3.weight: 20
transformer.layers.0.linear_net.3.bias: 20
transformer.layers.0.norm1.weight: 20
transformer.layers.0.norm1.bias: 20
transformer.layers.0.norm2.weight: 20
transformer.layers.0.norm2.bias: 20
output_net.0.weight: 20
output_net.0.bias: 20
output_net.1.weight: 20
output_net.1.bias: 20
output_net.4.weight: 100
output_net.4.bias: 100
```

[9]:
```python
for odict in w:
    print(odict)
    print(w[str(odict)])
    #print(str( (w[str(odict)][5])))
    print("-------------------------------------------------------------------------------")
```

```
input_net.1.weight
tensor([[-0.0426,  0.0830, -0.1355,  …,  0.0443,  0.0756, -0.1513],
        [-0.0113, -0.0625, -0.0121,  …,  0.0718, -0.0941,  0.0002],
        [-0.0040, -0.0080, -0.0790,  …, -0.0071, -0.0093, -0.0629],
        …,
        [-0.0118,  0.0790,  0.0068,  …, -0.0352,  0.0384, -0.0557],
        [ 0.0534,  0.0784,  0.0349,  …, -0.0320, -0.0679,  0.0778],
        [-0.0113, -0.0480,  0.0800,  …, -0.0385,  0.0781,  0.0467]],
       device='cuda:0')
-------------------------------------------------------------------------------
input_net.1.bias
tensor([-0.0159, -0.0223, -0.0355, -0.0940, -0.0297, -0.0430, -0.0663, -0.0205,
         0.0072,  0.0305, -0.0632, -0.0675, -0.0037, -0.0083,  0.0722, -0.0591,
         0.0398, -0.0565,  0.1008, -0.0285], device='cuda:0')
-------------------------------------------------------------------------------
transformer.layers.0.self_attn.qkv_proj.weight
tensor([[-0.2430, -0.3013, -0.0556,  …,  0.2517,  0.0555, -0.1104],
        [ 0.2999, -0.3773, -0.1769,  …, -0.1504,  0.2451, -0.1384],
        [-0.0391,  0.0728, -0.3161,  …, -0.1138,  0.0396, -0.1447],
        …,
```

```
        [ 0.0826,  0.2084, -0.1720,  …,  0.0579,  0.0331, -0.1127],
        [ 0.1491,  0.0381,  0.0450,  …, -0.2364,  0.2155, -0.0605],
        [ 0.2074, -0.0187, -0.2930,  …, -0.1836,  0.2611,  0.0813]],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.qkv_proj.bias
tensor([-1.9021e-02,  8.6855e-03,  1.8794e-02, -1.3951e-02,  4.9918e-02,
         1.2311e-02, -5.1049e-02, -1.5662e-02,  2.7849e-03, -2.3044e-02,
        -9.9867e-03, -2.5093e-02, -1.3689e-02,  1.1666e-02,  4.5271e-03,
         8.4966e-03,  4.2968e-03,  1.8027e-02, -3.6142e-02,  6.7878e-03,
        -1.0589e-05, -2.8760e-06, -3.2535e-06,  6.7715e-08,  6.6309e-06,
         2.0782e-06, -3.8687e-06,  5.5298e-07, -4.4638e-06, -6.5926e-06,
        -6.0436e-06, -8.6366e-06, -3.0002e-06, -1.7352e-06, -5.3702e-06,
        -8.2102e-06, -1.0488e-05,  1.0176e-06, -1.2994e-05, -3.7646e-06,
        -4.9714e-03,  1.3532e-02,  8.8236e-03, -1.6997e-03, -1.0845e-02,
         1.8569e-02,  2.5310e-02, -2.1919e-02, -1.2375e-02,  2.4160e-02,
         8.9144e-03, -6.1232e-03,  2.6090e-03, -1.8614e-02, -2.4896e-03,
         1.7779e-02,  1.9605e-02, -2.7597e-02, -1.8131e-02,  1.9189e-02],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.o_proj.weight
tensor([[ 3.9300e-02,  1.5430e-01, -1.6007e-01,  3.4046e-01, -8.0391e-02,
         -2.1460e-01, -2.6471e-02,  3.7541e-01,  2.9021e-01, -2.4221e-01,
          4.2790e-02, -1.0444e-02, -2.1112e-01, -1.0738e-01, -6.4027e-02,
         -2.2361e-02,  2.5545e-01,  3.8463e-01,  3.4966e-01, -1.0363e-01],
        [ 2.3965e-01, -2.6541e-01,  2.4026e-01, -4.7657e-01,  2.0091e-01,
         -3.5155e-01,  1.0467e-03,  8.3089e-03,  2.7669e-01,  1.5923e-02,
          4.2585e-01,  6.2842e-02,  2.5477e-02, -3.3990e-01, -1.9640e-01,
          7.4659e-02,  5.9900e-02, -6.0098e-02,  3.2198e-01, -1.0124e-01],
        [-1.7885e-01,  3.7859e-01,  1.9229e-01,  1.0083e-01, -3.7061e-01,
         -1.4214e-01, -1.8056e-02,  3.2494e-01,  1.8047e-01,  1.4467e-02,
          1.2526e-01, -1.2455e-01, -4.6346e-02,  3.2008e-01,  1.1045e-01,
         -3.3875e-01,  1.4276e-01, -4.0920e-01, -1.9191e-01,  1.3537e-03],
        [-2.1864e-01, -2.4431e-01, -2.2009e-01,  4.7809e-01, -3.2726e-01,
         -2.4538e-01,  3.1906e-01, -2.3436e-01,  1.6917e-01,  2.6328e-01,
          4.3411e-02, -5.0883e-02, -3.8802e-01, -2.4993e-01,  1.8766e-01,
         -1.5252e-01, -7.6675e-02,  2.6079e-01,  2.2735e-01, -2.8642e-01],
        [-2.6264e-01, -1.0817e-01, -2.4506e-01,  2.1671e-01,  1.1250e-01,
          1.3437e-01,  2.7508e-01,  2.0964e-01,  3.6131e-01,  7.7473e-02,
         -3.4030e-01,  1.8101e-01,  1.8413e-01,  3.9292e-01,  2.8767e-01,
         -3.1206e-01, -6.2611e-02,  7.1606e-02,  5.1040e-02, -2.3593e-01],
        [-2.6070e-02, -1.3600e-02,  7.8285e-02, -1.2425e-01,  1.7630e-01,
         -2.9765e-01,  4.7877e-03,  9.9838e-02,  1.5677e-01, -2.5949e-01,
         -3.7723e-02, -2.1371e-01, -1.6365e-01,  4.1497e-02, -3.5694e-01,
          4.1300e-03,  2.0412e-01,  1.3795e-01,  3.0513e-01,  1.2166e-01],
        [-1.2877e-01, -1.6657e-01,  2.6370e-01,  2.3949e-01,  2.3025e-01,
          4.6043e-02,  3.3129e-01,  2.5118e-01, -2.0080e-01,  2.0015e-01,
          9.3169e-03, -2.3361e-01,  2.6346e-01, -3.9239e-01, -3.9309e-02,
```

```
  5.5856e-02,  2.7670e-01,  2.7675e-01, -2.5129e-02, -4.0772e-02],
[ 2.2870e-01, -3.8884e-01, -2.5483e-01,  3.6218e-01,  1.7798e-01,
 -2.5410e-01, -1.9862e-01, -2.1763e-01, -2.9794e-01, -1.7372e-01,
  1.5462e-02,  1.9650e-01,  1.1324e-01,  1.2491e-01,  3.8530e-01,
  2.5048e-01, -4.1420e-01, -1.2719e-01,  7.4471e-02, -2.7183e-01],
[ 2.6579e-01,  7.1973e-02, -4.2794e-02, -1.9007e-01,  7.6241e-02,
 -7.8151e-02, -9.1427e-03,  4.0073e-01, -3.2479e-01,  1.1509e-01,
 -2.9960e-01,  8.6244e-02, -3.4551e-01, -2.9932e-01, -2.0352e-02,
  1.2062e-01,  1.8492e-01, -5.9112e-02, -2.3800e-01, -2.6087e-01],
[-2.3152e-01, -9.7577e-02, -1.6640e-02, -3.4620e-01,  3.1135e-01,
  2.8519e-01, -1.7956e-01,  2.1645e-01,  1.0864e-02,  1.5852e-04,
 -1.1808e-01, -3.4822e-01,  2.7803e-01,  2.1170e-04,  2.2106e-01,
 -3.5171e-01,  2.9661e-01, -1.6026e-01,  5.7542e-02,  1.1113e-01],
[-5.4790e-02, -3.1899e-01,  3.4522e-01,  3.5711e-01, -2.8808e-01,
  9.9178e-02, -2.1141e-01,  1.3284e-01, -1.3214e-01, -2.9922e-01,
 -3.0251e-01,  2.5096e-01,  2.7822e-01,  7.4604e-02,  8.6583e-02,
 -1.4806e-01, -4.1878e-01, -2.3661e-02, -3.3270e-01, -2.6812e-01],
[ 7.7257e-02, -2.4553e-01, -4.4820e-02, -8.4987e-02,  2.7773e-01,
 -1.1045e-01,  1.6847e-01, -3.2037e-01,  1.9212e-02,  9.6020e-02,
  1.6728e-01, -2.0853e-01, -3.4596e-03, -2.1334e-01, -3.1283e-01,
 -9.7962e-02,  2.1000e-01,  5.1625e-02, -3.8537e-01, -3.8122e-01],
[ 9.0424e-02, -9.6286e-02, -3.3184e-01,  3.4581e-01, -6.2467e-02,
 -1.9548e-01, -9.2542e-02,  2.9392e-01, -1.4801e-01,  1.7798e-01,
 -6.8696e-02, -2.0268e-01, -1.6288e-01,  3.6107e-01,  4.1237e-01,
 -2.3723e-03,  2.6720e-01,  1.3891e-01, -1.2396e-01,  2.1990e-01],
[-3.6695e-01,  3.0963e-01, -4.2472e-01,  5.1374e-01, -1.3238e-01,
  3.8991e-01,  2.6568e-01, -2.9578e-01, -2.0840e-01,  1.7463e-01,
 -1.4375e-01, -1.5553e-01, -3.6374e-02, -2.3500e-01,  5.5979e-02,
  3.4521e-01,  3.0242e-01, -3.8835e-01, -3.1342e-01,  2.2468e-01],
[ 3.2081e-01, -3.2155e-01,  1.1298e-01, -7.1448e-02, -3.2710e-01,
 -9.4118e-02,  2.9932e-01, -2.1493e-01, -2.6783e-01,  1.0166e-01,
  2.1740e-02,  4.9695e-02, -3.0826e-01, -4.9929e-02,  3.1460e-01,
  1.7469e-01,  2.4459e-01, -2.1085e-01,  1.5810e-01,  3.3129e-01],
[ 3.1080e-01,  7.3185e-03,  1.2435e-02, -2.1366e-01,  9.0454e-02,
  1.8954e-01,  2.8447e-01, -1.8403e-01, -9.0546e-02,  3.6572e-01,
  2.7210e-01, -9.2192e-02,  2.1218e-01,  9.9962e-02,  5.7437e-02,
 -1.9162e-02,  2.2503e-01,  1.4341e-01, -4.0602e-01,  2.1000e-01],
[-2.3323e-01, -2.6807e-01, -3.1373e-01,  8.4128e-02, -2.1384e-01,
 -2.1553e-01,  3.9431e-01,  8.1375e-02,  1.0152e-01, -2.6921e-01,
 -4.6624e-01,  3.8389e-01, -6.4737e-02, -6.2910e-02,  3.6828e-01,
  2.6006e-01, -1.0903e-01,  2.8182e-01,  3.8593e-02,  6.7008e-02],
[-9.4465e-02,  3.2956e-01,  1.7235e-01, -5.9185e-01, -4.5729e-02,
  1.8897e-01, -3.2109e-01, -1.4826e-01, -2.6040e-01, -3.4166e-01,
  2.9055e-01, -2.5409e-01, -2.4913e-01, -2.0104e-01, -3.4673e-01,
 -2.4320e-01,  2.6744e-01,  3.9724e-01,  2.5762e-01,  1.3068e-01],
[ 3.1442e-01,  1.6499e-02,  3.0298e-02, -6.6914e-01,  8.4799e-02,
 -5.8279e-02,  1.3262e-01, -5.0273e-02,  2.7834e-01,  2.9746e-01,
 -2.9405e-01, -2.5347e-02, -2.7366e-01,  3.9936e-01, -2.4032e-01,
```

```
          3.7606e-01, -1.1691e-01, -1.3501e-01, -2.6152e-01, -1.1370e-01],
        [ 1.0719e-01, -3.0628e-02, -3.8295e-01,  3.8105e-03,  1.2373e-01,
         -3.0250e-01, -2.4089e-01, -9.8255e-02,  6.1202e-02,  3.4960e-01,
         -2.8705e-01, -1.8721e-01,  3.5577e-01,  1.8840e-01,  3.8080e-02,
          8.0315e-02, -5.0535e-02, -4.3782e-01,  2.7658e-01, -2.6651e-01]],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.o_proj.bias
tensor([ 0.0143,  0.0014,  0.0151, -0.0124, -0.0174, -0.0224,  0.0139, -0.0177,
        -0.0136, -0.0031, -0.0063,  0.0046, -0.0251,  0.0217,  0.0240,  0.0169,
         0.0079, -0.0048, -0.0020,  0.0026], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.linear_net.0.weight
tensor([[ 0.0220, -0.1582,  0.0207, -0.0474,  0.1898, -0.0128,  0.0815,  0.0648,
          0.1851,  0.0276, -0.0515, -0.0819,  0.0184,  0.1887,  0.0533, -0.0239,
         -0.1374, -0.1760,  0.0007,  0.1380],
        [ 0.1435, -0.1452,  0.0044,  0.0776, -0.0307,  0.1070, -0.2328, -0.2288,
          0.1080, -0.1967, -0.0400, -0.1591, -0.2037, -0.1655, -0.1684,  0.0354,
         -0.1693,  0.0255, -0.0254, -0.1672],
        [ 0.1914, -0.2605, -0.1072, -0.0850,  0.0454, -0.1618,  0.0108,  0.1154,
         -0.1506,  0.0445,  0.0828,  0.1549, -0.0935, -0.1520,  0.1437,  0.0809,
         -0.1932,  0.0601,  0.1559, -0.2115],
        [-0.0611,  0.2040,  0.0073, -0.0927,  0.2024, -0.0902,  0.1778, -0.0139,
          0.0155,  0.0216, -0.0089, -0.0621,  0.0704, -0.0301, -0.0305,  0.1998,
          0.2092,  0.1009, -0.1116,  0.0463],
        [ 0.1706,  0.0110,  0.1012, -0.2357, -0.1409,  0.0434,  0.2049,  0.2532,
          0.0495, -0.1375,  0.1011, -0.1235, -0.0725,  0.1801, -0.1188, -0.0185,
          0.1476, -0.0963, -0.1243,  0.0305],
        [-0.0283, -0.0153,  0.2354,  0.0600,  0.1959,  0.0925, -0.0665, -0.1798,
         -0.1408,  0.2850, -0.1438, -0.0271, -0.2547, -0.0669,  0.0045, -0.1001,
          0.0369,  0.1848,  0.1472, -0.0725],
        [ 0.0998,  0.0873,  0.2016, -0.1315, -0.0708, -0.1980, -0.1721, -0.2271,
          0.1340,  0.0053,  0.0721,  0.1204,  0.1823, -0.0028,  0.0389, -0.0198,
         -0.2284,  0.1866,  0.0380,  0.0288],
        [ 0.1206, -0.1299,  0.0502,  0.0863, -0.0168,  0.0961, -0.0646,  0.1639,
          0.0680,  0.0260,  0.0373,  0.2075, -0.0865,  0.0152, -0.1084,  0.1103,
          0.1927, -0.1556, -0.0762,  0.0010],
        [-0.0453,  0.2345,  0.0609,  0.1260,  0.1580, -0.0338,  0.0818, -0.0718,
          0.0411,  0.0799, -0.1617,  0.1566, -0.1592, -0.0258, -0.0206,  0.0139,
          0.1882, -0.2018, -0.0196,  0.1142],
        [ 0.0611, -0.1771,  0.1371,  0.1328, -0.1347, -0.0497,  0.1758, -0.0782,
          0.0204,  0.2318, -0.0948, -0.1120,  0.2005,  0.0769, -0.0447, -0.1561,
          0.0951,  0.0750, -0.1414, -0.1385],
        [ 0.1369, -0.1503, -0.2506, -0.0709, -0.1235, -0.0210, -0.1313,  0.1183,
         -0.0039, -0.0568,  0.2446, -0.0901, -0.1918, -0.2145, -0.1749,  0.1546,
         -0.1130, -0.0292,  0.0128,  0.1491],
        [ 0.0346, -0.0816,  0.2127,  0.0656,  0.1302, -0.0807,  0.0217, -0.0105,
          0.1348, -0.1118,  0.1424, -0.1706,  0.0416, -0.0420, -0.0870,  0.2184,
```

```
 -0.2399,  0.0864, -0.0689,  0.0023],
[-0.1292, -0.1715, -0.1902, -0.2051,  0.0750, -0.0805, -0.1526,  0.1083,
 -0.1916, -0.1979, -0.1352,  0.2015, -0.0299,  0.2124,  0.2220, -0.1711,
  0.1993,  0.0871,  0.2007,  0.1539],
[ 0.1012, -0.0464,  0.2094, -0.1383, -0.2342, -0.1700, -0.0585,  0.0642,
  0.1082, -0.1822, -0.1884, -0.0262, -0.0051, -0.0082, -0.0995, -0.1765,
  0.0904, -0.0259, -0.0949,  0.1545],
[ 0.0328, -0.0910,  0.1535,  0.0393,  0.1358, -0.1689, -0.1375,  0.1750,
  0.1895, -0.1974, -0.1742, -0.1844,  0.0670,  0.0155, -0.0813, -0.0605,
  0.1981, -0.1676,  0.0907, -0.0912],
[-0.1224,  0.2179,  0.1114, -0.2129, -0.0698, -0.0966, -0.0563,  0.0979,
 -0.0921, -0.0457,  0.1399, -0.0044, -0.1772, -0.2176, -0.0730,  0.0339,
 -0.0281,  0.2007, -0.1894, -0.0812],
[-0.1426, -0.1228,  0.0060, -0.2390, -0.1782, -0.0465,  0.0495,  0.1043,
 -0.1535,  0.1595,  0.1045, -0.1081, -0.0696,  0.0338, -0.0548, -0.1596,
 -0.1260,  0.0475, -0.1597,  0.1502],
[ 0.2058,  0.0498,  0.2106, -0.1290, -0.1922,  0.1544,  0.1204, -0.1484,
 -0.2368, -0.0588,  0.0659, -0.0685, -0.0941,  0.1459, -0.1971,  0.1730,
 -0.1363,  0.0076,  0.0475, -0.1335],
[-0.1255, -0.1566,  0.1030, -0.1913, -0.2174, -0.1812,  0.1427,  0.1253,
  0.1768,  0.0928,  0.1041,  0.1995, -0.0792, -0.0022,  0.0561,  0.2482,
 -0.1832,  0.0623,  0.0892, -0.1486],
[-0.0346, -0.1273, -0.1001, -0.1947,  0.0352,  0.2440,  0.1030,  0.0143,
  0.0084, -0.1575, -0.1190,  0.0165,  0.1750,  0.2302, -0.1278,  0.0046,
 -0.1232,  0.1009,  0.1397,  0.1731],
[-0.0682,  0.1915,  0.0290,  0.1441, -0.0324, -0.1034, -0.2253, -0.0042,
 -0.0246,  0.1582, -0.0199, -0.0802, -0.1164,  0.1530, -0.1447, -0.1258,
  0.2173, -0.0863, -0.1847, -0.2268],
[ 0.0795, -0.2166, -0.0490,  0.1736,  0.1743,  0.1497,  0.1394, -0.0256,
  0.2037, -0.1481, -0.0898, -0.0916,  0.2254, -0.0664,  0.1790,  0.0559,
  0.0638, -0.0016, -0.0816, -0.1273],
[ 0.0626,  0.1112, -0.1439, -0.1089, -0.1743,  0.2241,  0.1036,  0.0780,
 -0.0729,  0.0155,  0.1699, -0.0409,  0.2140,  0.2178, -0.0907,  0.1116,
  0.1011, -0.0238, -0.0629,  0.1966],
[-0.1917, -0.0353, -0.1378, -0.0088, -0.0073, -0.1723,  0.1476,  0.0494,
  0.1487,  0.1039, -0.0770,  0.1385, -0.1170,  0.0079, -0.0677, -0.1717,
 -0.0761, -0.0729, -0.1685,  0.1919],
[-0.0354,  0.0346,  0.2048,  0.0181, -0.1546,  0.0078, -0.2380,  0.0538,
 -0.0079,  0.1931, -0.0859,  0.0785, -0.1484, -0.0224, -0.0650,  0.1291,
 -0.0670,  0.0274,  0.2084, -0.1688],
[ 0.0152,  0.0177,  0.0978, -0.1081, -0.0893, -0.1864,  0.0531, -0.0971,
 -0.2035, -0.1125, -0.0182, -0.0614, -0.1301, -0.1589,  0.1648,  0.0185,
  0.0862, -0.1118, -0.0575,  0.1908],
[ 0.2149,  0.1449, -0.1611,  0.0740, -0.0893,  0.1355,  0.1669, -0.2103,
  0.0425, -0.1682,  0.1731, -0.0349,  0.1265,  0.1044,  0.1362, -0.0888,
  0.1614,  0.1342, -0.0797, -0.0808],
[-0.2657,  0.0739,  0.0571,  0.1027,  0.0198,  0.0597,  0.1572, -0.1548,
  0.0586, -0.0785,  0.0078, -0.0518, -0.0069,  0.2225, -0.1901,  0.1412,
```

```
          0.1556,  0.1726, -0.0625, -0.1266],
        [-0.0017,  0.1516, -0.2224,  0.0156, -0.1911,  0.0881,  0.1929, -0.0981,
         -0.1244,  0.2277, -0.1296,  0.1151,  0.1244,  0.1136, -0.1915,  0.1522,
         -0.2048, -0.0071,  0.0448,  0.1227],
        [-0.1477, -0.0290,  0.1818,  0.1782, -0.1990,  0.0656,  0.2150,  0.0015,
         -0.1774,  0.2158, -0.0827, -0.1883,  0.1252, -0.1703, -0.1927, -0.0962,
          0.1216, -0.0350, -0.0039,  0.1146],
        [-0.0836, -0.0474, -0.1221,  0.1633,  0.0148,  0.0744,  0.0189,  0.0270,
          0.1528, -0.1995,  0.1837,  0.1109,  0.0857,  0.0337,  0.0870, -0.0229,
         -0.1800,  0.0208,  0.0780,  0.1907],
        [ 0.1976, -0.2071,  0.2075,  0.0348,  0.0665,  0.1313,  0.0196,  0.1456,
         -0.2641,  0.0530, -0.2606, -0.1759, -0.2035,  0.1486, -0.0035, -0.1912,
         -0.2470,  0.1282,  0.0781,  0.1162],
        [-0.1088, -0.0962,  0.0392,  0.1168,  0.1591, -0.1559,  0.0527, -0.2426,
         -0.1235, -0.2154, -0.1626, -0.1810, -0.1233,  0.0746, -0.1855,  0.1583,
         -0.1327,  0.0969, -0.0690, -0.1055],
        [-0.0568,  0.0322,  0.2254, -0.0317,  0.0417, -0.1366, -0.1358,  0.1062,
          0.2596,  0.0131,  0.0527, -0.0602, -0.2092,  0.1239,  0.0498, -0.1376,
          0.1773, -0.2078, -0.1166, -0.0856],
        [-0.0977, -0.1706,  0.0588,  0.0855,  0.0807, -0.0043,  0.1330, -0.1142,
          0.1528,  0.1203,  0.0782, -0.0802, -0.1394, -0.1516, -0.2005, -0.1934,
         -0.1367, -0.0665, -0.0530,  0.0340],
        [ 0.0600, -0.2498,  0.0739,  0.0999, -0.1617,  0.1041,  0.1422, -0.1257,
          0.0403,  0.0068,  0.0087,  0.1309, -0.1638, -0.0147,  0.0176,  0.0897,
         -0.0188, -0.0438, -0.1927,  0.0857],
        [-0.2279, -0.0381, -0.2422,  0.1774,  0.1196, -0.2081,  0.0257, -0.1733,
         -0.0098,  0.2231, -0.0406, -0.0618,  0.1227,  0.0588,  0.1456,  0.2117,
          0.0600, -0.1392, -0.1111,  0.0420],
        [-0.0214,  0.2124,  0.1912, -0.1696,  0.2361,  0.0011, -0.0304,  0.0023,
         -0.1696, -0.1923, -0.1784, -0.1397, -0.2063, -0.0284, -0.2200,  0.1091,
          0.2016, -0.1816, -0.1688, -0.2076],
        [-0.0025, -0.1754, -0.0820, -0.1946,  0.1067,  0.2083,  0.1736,  0.0666,
          0.0785,  0.0449,  0.1810,  0.1321, -0.1455, -0.1131, -0.2015,  0.1505,
         -0.1003, -0.1736, -0.0197, -0.0086],
        [-0.1471,  0.0094, -0.1277, -0.0698, -0.0131, -0.2653,  0.1815, -0.0844,
          0.2165,  0.0534, -0.1822, -0.0312,  0.1325, -0.0185,  0.2136,  0.2273,
         -0.2567, -0.1970,  0.0924, -0.0682]], device='cuda:0')
------------------------------------------------------------------------------
transformer.layers.0.linear_net.0.bias
tensor([-0.2083, -0.0877,  0.1062,  0.1710,  0.2065,  0.2152, -0.0104, -0.0035,
         0.0620, -0.0553,  0.0427, -0.1329, -0.0666,  0.0624, -0.1101, -0.1081,
         0.0679, -0.1865,  0.0149, -0.1978, -0.2396, -0.0805,  0.0572, -0.0583,
         0.1175, -0.1589,  0.0676,  0.2155, -0.1460, -0.1197, -0.2007,  0.1704,
         0.2037,  0.1197,  0.0245,  0.2344, -0.0937,  0.2188,  0.1511, -0.0686],
       device='cuda:0')
------------------------------------------------------------------------------
transformer.layers.0.linear_net.3.weight
tensor([[-2.3248e-02, -1.4003e-01,  6.7980e-02,  1.0058e-01,  2.9405e-02,
```

```
 -7.4421e-02, -1.1073e-01, -1.1528e-01,  1.6050e-01,  1.3352e-01,
  2.3826e-02, -7.7849e-02, -1.8912e-02, -1.0085e-01, -1.0664e-01,
 -4.5693e-02, -9.0563e-02, -2.6136e-02,  5.5647e-02,  1.3810e-01,
  9.7700e-02, -9.3831e-02, -1.0833e-01,  4.4943e-02, -8.3746e-02,
  1.2479e-01, -6.9906e-02,  6.2085e-02, -3.2759e-03,  9.3731e-02,
 -5.5549e-02, -6.7032e-02, -3.0474e-03, -4.6413e-02, -4.2102e-02,
  1.5585e-01,  1.7016e-01, -1.6333e-01, -2.9108e-02,  1.7239e-01],
[ 1.2656e-01,  1.7612e-02,  1.6468e-01, -1.0386e-01,  4.9900e-02,
  1.3289e-01,  2.8935e-02, -5.5856e-02, -1.9847e-01,  4.3963e-02,
 -5.9959e-02,  7.5695e-02,  1.1000e-01, -1.7853e-01, -8.9049e-02,
  8.6549e-02,  7.9422e-02,  9.1492e-02,  1.5912e-01, -2.9033e-02,
  1.0453e-01, -3.2547e-02, -7.6230e-02, -5.9919e-02,  2.0133e-02,
  1.0771e-01, -1.2389e-01,  1.4524e-01, -3.0577e-02, -1.1822e-02,
  1.1511e-01, -3.7428e-02, -4.3950e-02, -1.8969e-01,  1.3545e-01,
  1.1720e-01,  5.8408e-02, -6.4763e-02,  1.1647e-01, -4.9178e-02],
[ 1.1510e-01, -1.3349e-02,  2.1403e-02,  1.6811e-01, -6.1787e-02,
 -2.1006e-01,  7.8230e-02, -1.3420e-01, -1.1154e-01, -1.5597e-01,
 -1.4157e-01, -1.3835e-01,  4.5284e-04, -1.0309e-01,  6.8354e-02,
  1.5216e-02,  6.1849e-02,  1.9031e-02, -1.9416e-02,  1.1530e-01,
 -1.0933e-01,  9.0659e-03, -5.7787e-02, -6.1292e-02, -1.1166e-01,
 -2.9734e-02, -1.2610e-01,  1.4948e-01,  1.0692e-01, -5.6729e-02,
  1.8953e-01, -9.6110e-02,  6.8510e-02,  1.1286e-01, -1.5527e-01,
 -1.3537e-01,  2.3654e-01, -8.0328e-02, -1.3012e-01, -6.1771e-02],
[ 3.1342e-02,  5.7512e-02,  2.9724e-02,  2.5530e-02,  1.8873e-01,
 -1.1517e-02, -5.1145e-02,  5.9292e-02,  3.2999e-02,  7.2946e-02,
 -3.0374e-02,  9.5705e-02,  2.7542e-02, -1.1080e-01,  1.4378e-01,
 -1.0835e-01,  9.2581e-02,  3.4244e-02, -1.4840e-02,  3.2247e-02,
  7.6818e-02, -1.2682e-01,  1.5947e-01,  2.8933e-02,  2.5350e-02,
 -1.2577e-01, -6.9784e-02,  1.1030e-02, -1.5235e-01, -8.6105e-02,
  7.6917e-02, -4.0159e-02, -2.1683e-02, -5.8952e-02,  1.0535e-01,
  2.9407e-02, -3.1499e-02,  1.3358e-01, -7.3469e-02, -2.2452e-02],
[ 4.1946e-02,  3.3784e-02,  8.4675e-02,  4.5385e-02,  1.1242e-02,
  1.5570e-01, -1.0597e-01, -1.9374e-02, -1.6775e-01,  3.3274e-02,
 -4.3739e-02,  1.3484e-01, -1.8237e-01,  5.9334e-02,  1.3811e-01,
 -8.3596e-02, -8.3386e-02, -1.2016e-02,  4.5653e-02,  9.6428e-02,
  6.9732e-04,  1.1063e-01,  8.8061e-03,  9.6510e-02,  5.9806e-02,
  4.2668e-02, -1.2674e-01, -3.8113e-02, -3.8736e-02,  3.3985e-02,
  9.8169e-02, -2.4458e-02,  4.9585e-02,  1.6429e-01, -6.2295e-02,
 -3.8984e-03, -1.2729e-01, -9.4391e-02, -1.8108e-01, -1.7208e-01],
[ 1.5597e-02, -3.6119e-02,  5.6970e-02,  7.6480e-02, -3.2835e-02,
  4.4982e-02,  1.1295e-01, -3.7698e-02,  1.2360e-02,  1.2975e-01,
  1.1987e-01,  6.7610e-02, -1.4461e-01,  1.7783e-02,  1.5536e-02,
 -1.1385e-01, -1.3964e-01,  6.0265e-02,  1.8968e-03, -3.3667e-03,
 -2.2594e-02, -1.0265e-01, -1.1295e-01, -1.0225e-03, -1.2421e-01,
 -6.4122e-02,  9.6073e-02, -1.5400e-01,  9.0940e-02,  4.9226e-02,
  1.0961e-01, -1.8324e-01,  1.0952e-01,  4.7556e-02, -5.3818e-02,
 -1.8722e-01,  6.1598e-02, -8.9453e-02, -1.3154e-01,  1.1950e-01],
[-1.1439e-01,  1.2539e-01,  7.2052e-02, -1.1954e-01, -1.5932e-01,
```

```
  1.1601e-01,  5.5858e-03,  1.3171e-01,  6.4289e-02,  1.3260e-01,
  5.4016e-02, -6.4578e-03, -1.4356e-01, -1.5832e-01,  4.5766e-02,
  3.9686e-02, -3.0410e-02, -9.0163e-02,  1.2015e-01,  1.7230e-01,
  1.4073e-01, -1.3326e-01, -5.2866e-02, -4.9984e-02,  1.1756e-01,
  1.1423e-01,  1.2451e-01, -1.9386e-02, -7.8859e-02, -5.2115e-02,
 -5.4028e-03,  9.2731e-02,  9.6771e-02, -8.6218e-02,  1.2475e-01,
  5.7153e-02,  9.7065e-02,  4.4109e-02, -1.4011e-01, -1.2295e-01],
[-1.0764e-01, -4.5221e-02,  8.8316e-02, -9.6532e-03, -6.2003e-03,
 -1.1687e-01,  7.2859e-02, -9.5183e-02, -1.0041e-02,  1.3636e-01,
  5.7827e-02,  1.2654e-01, -1.2534e-01,  1.0963e-01, -7.5847e-02,
  1.3997e-01, -1.5720e-02, -1.7487e-01,  7.6876e-02,  9.8009e-02,
  3.4839e-02, -3.4899e-02, -2.5214e-02, -1.2762e-01, -1.6603e-01,
 -3.3467e-02, -6.4587e-02, -1.0537e-02,  4.1872e-02,  3.6287e-04,
  1.1049e-01, -1.9356e-01,  4.3206e-02, -1.1159e-01,  2.3539e-02,
 -1.3288e-01,  8.0649e-02, -8.9560e-02,  1.1247e-01,  1.5508e-01],
[ 5.7574e-02, -1.8207e-03,  7.6143e-02,  8.6898e-02,  2.2067e-05,
 -1.8189e-01,  2.6879e-02,  4.4467e-02,  1.1670e-01,  4.1375e-02,
 -1.3487e-01, -7.3721e-02,  5.5188e-02,  7.5983e-02,  6.9299e-02,
  7.5366e-02, -1.0929e-01, -1.9294e-02,  7.2284e-02, -1.6521e-01,
 -6.7284e-02, -2.4902e-02,  3.9954e-02,  2.2884e-02,  1.0136e-02,
  4.8566e-02,  3.0589e-02, -9.4020e-02,  1.3257e-02, -4.6145e-02,
 -1.2254e-01,  4.8716e-02,  1.1287e-01,  4.0949e-02,  6.0800e-02,
 -1.4294e-01,  8.2695e-02, -1.0799e-01, -8.6779e-02,  1.2256e-01],
[-7.8545e-02,  5.4670e-03, -1.1806e-01, -1.5237e-01, -1.6393e-02,
  1.9851e-01,  1.2862e-01, -5.7195e-02,  1.0510e-01,  1.2216e-01,
 -1.2094e-01, -1.3092e-02,  2.4223e-02, -3.8890e-02,  3.9548e-02,
 -4.3339e-02,  3.8736e-02,  1.3141e-01,  1.0224e-01,  1.3053e-01,
 -1.5459e-01,  3.8113e-02,  5.4088e-02, -1.3933e-02,  1.8965e-01,
  3.9376e-03, -1.0528e-01, -3.7259e-03,  8.1795e-02,  3.7377e-03,
  7.1092e-02,  1.1193e-01,  1.1380e-01, -9.5490e-02, -1.2014e-01,
 -1.1399e-01, -1.1637e-01, -5.6871e-02, -6.5946e-02,  7.0625e-02],
[-1.6848e-01,  5.9177e-02, -3.7649e-02,  5.5298e-02,  4.0399e-02,
  1.8512e-01, -1.6183e-01, -1.8777e-02, -3.8357e-02, -4.6416e-02,
  1.8125e-02,  5.8279e-02, -1.7110e-01, -8.3932e-02, -1.6990e-02,
 -3.8576e-02, -1.0556e-01,  1.8054e-01, -1.3908e-01,  5.1388e-02,
  1.4072e-01, -1.5272e-01, -1.3105e-01, -9.6151e-02,  1.1269e-01,
 -1.1768e-01,  8.8635e-02,  8.6237e-02, -5.7701e-02, -4.4256e-02,
  9.0525e-02, -5.1036e-02,  1.1950e-01,  1.5076e-01, -1.1237e-01,
 -9.7689e-02,  1.1094e-01, -4.9394e-02,  1.9688e-02,  5.4758e-02],
[ 8.7793e-02,  9.3800e-02,  3.5583e-02, -5.2501e-02, -9.2548e-02,
 -5.1854e-02, -1.2920e-01, -8.8632e-03,  5.7606e-02,  2.4180e-02,
  2.5847e-02,  1.2860e-01, -7.6999e-02,  1.0584e-01, -7.8120e-02,
 -9.5864e-02, -9.4234e-02, -8.0790e-02,  1.7018e-01,  2.9638e-02,
  5.9762e-02,  1.3451e-01,  1.2964e-01, -6.6891e-03, -1.2486e-01,
  2.5110e-02,  1.4936e-01,  1.1940e-01,  1.1325e-01, -9.3430e-02,
 -5.6965e-02,  6.3051e-02,  1.3084e-01, -8.2951e-03, -8.9393e-02,
  5.7463e-02,  1.0821e-02,  7.3732e-02,  1.2135e-01,  5.8555e-02],
[ 8.3509e-02, -1.3480e-02,  2.1922e-02, -3.2583e-02, -7.8745e-02,
```

```
 -1.4153e-01,  1.4234e-01, -7.9618e-02,  1.3685e-02, -8.9479e-02,
  3.1697e-02, -1.6636e-01, -3.9868e-02, -5.3493e-02, -3.0812e-02,
 -1.4581e-01,  6.1576e-02, -9.3283e-04, -1.0650e-01, -9.3950e-02,
  1.4488e-01,  7.1250e-02, -3.9858e-02, -3.3888e-02,  8.5378e-02,
 -3.1384e-02, -7.3202e-02, -1.8173e-01,  8.7088e-02, -8.2157e-02,
 -7.4840e-02,  1.2624e-01,  2.6377e-02, -9.9578e-02,  6.5744e-02,
 -1.2868e-01,  1.1960e-02,  1.2932e-01, -1.5727e-01,  5.0051e-02],
[-4.9482e-02,  1.4978e-02, -1.1497e-02,  4.9734e-02, -8.6194e-02,
 -3.6785e-02,  5.6190e-02,  9.0005e-02, -3.6276e-03,  1.4052e-01,
  1.0470e-02,  1.5017e-01, -2.0742e-02,  1.6135e-02,  4.1408e-02,
 -1.2259e-01,  6.6537e-02, -1.4863e-02,  1.0925e-01,  7.5860e-02,
 -1.1099e-01,  1.1443e-01, -1.2294e-01,  4.1695e-02, -3.4226e-03,
 -1.0743e-01,  3.3644e-02, -1.0720e-01, -5.9030e-02,  1.4081e-02,
  9.5060e-02,  8.7470e-03,  1.5558e-01,  6.7342e-02,  4.3099e-02,
 -6.3244e-02,  1.0704e-01, -3.5460e-02,  4.6110e-03,  6.1051e-02],
[ 4.2553e-02,  1.2222e-01,  1.3396e-01, -7.6258e-02,  6.4322e-02,
  1.8662e-01,  5.7588e-02,  6.4256e-02, -1.2046e-01,  1.1765e-01,
  2.3899e-02, -1.2917e-01, -1.1470e-01,  1.2473e-01,  5.0470e-02,
 -5.2394e-02,  4.3036e-02,  6.4223e-02, -1.2402e-01,  2.2617e-02,
  1.5842e-01,  1.0402e-01,  3.0186e-02,  2.6230e-02, -6.5914e-03,
 -9.6124e-02,  5.0626e-02,  1.5985e-01, -6.9592e-02,  2.8067e-02,
 -3.4733e-02,  2.6654e-02,  1.1356e-01,  5.1715e-02,  9.8316e-02,
  1.4517e-01, -1.4267e-01, -2.4873e-02,  1.4922e-01, -9.8187e-02],
[ 7.7477e-02, -1.3774e-01,  9.7658e-02, -6.8949e-02, -3.6461e-02,
  1.9963e-02,  6.3633e-02, -1.1309e-01, -1.8877e-01, -9.7785e-02,
  1.0354e-01,  1.6721e-01, -1.0887e-01,  2.6476e-02,  9.0220e-02,
  9.4995e-02, -1.4114e-01, -1.2160e-01,  9.0959e-02, -7.2539e-02,
 -3.3605e-02, -3.3377e-02,  1.3811e-01,  8.9561e-02, -7.9316e-02,
 -3.3912e-02, -1.0690e-01, -3.0268e-02,  6.8768e-02, -1.4742e-02,
  1.5489e-01,  9.3560e-02, -6.3252e-02, -3.6845e-02, -1.1042e-01,
  5.6352e-02,  4.5003e-02,  1.3543e-01, -1.3146e-02,  5.7012e-02],
[-1.2628e-01,  1.0503e-01,  4.5043e-02,  1.3322e-01,  1.3146e-01,
 -1.4552e-01,  4.2094e-03, -7.3635e-02, -1.1159e-02,  7.5926e-02,
  7.3516e-02, -6.9853e-02, -5.9753e-03, -1.0128e-02, -1.3715e-01,
  3.9053e-02,  1.5866e-02,  1.3811e-01, -1.1836e-01, -8.6133e-02,
 -1.1643e-02,  1.0156e-01,  7.6735e-02, -3.8489e-02, -3.3364e-02,
  1.8007e-02, -1.3383e-01, -1.0622e-01, -1.6856e-01,  1.4299e-01,
 -1.3122e-01, -5.2532e-02,  1.0375e-02,  1.3439e-01, -7.4160e-02,
 -6.8667e-02,  7.9113e-02,  4.8892e-02, -7.2388e-02,  1.7635e-02],
[ 6.5443e-02,  1.6236e-02, -4.4680e-02, -8.0306e-02, -1.5033e-02,
 -2.2162e-02,  1.9270e-01,  1.9006e-02,  1.2121e-01, -1.2193e-01,
  1.5475e-02,  1.3543e-01,  8.3120e-02,  5.9455e-03,  1.2986e-01,
  1.1216e-01,  1.5433e-01,  3.0940e-02,  8.1724e-02, -1.5509e-01,
 -5.7629e-02,  1.1517e-02,  5.5820e-02, -1.1642e-01,  9.7611e-02,
  1.1640e-01, -2.8629e-02, -1.9236e-03, -3.4105e-02, -1.3160e-01,
  1.2331e-01, -6.4095e-02,  8.6884e-02,  4.4776e-02,  1.0023e-01,
 -5.7664e-02,  8.8626e-02, -5.4667e-02, -1.4338e-01,  1.7699e-01],
[ 3.9553e-02,  1.8608e-01,  7.8487e-02,  7.6896e-03, -6.9675e-02,
```

```
          1.5773e-02, -1.5697e-02, -1.3080e-01, -4.1601e-02,  1.3809e-02,
         -5.3972e-02,  7.1123e-02,  6.8045e-02,  2.3277e-03,  1.4762e-02,
          1.5674e-01,  1.1021e-01,  1.3749e-01, -5.4157e-02, -6.9928e-02,
          1.2688e-01,  8.9706e-02, -1.6618e-01,  8.7206e-02,  3.2163e-02,
         -1.4275e-01,  1.5147e-01, -1.3736e-01,  8.2404e-02,  9.1387e-02,
         -1.6246e-01,  4.4661e-02, -6.1852e-02, -1.1094e-01,  1.7061e-01,
         -5.1196e-02, -7.4550e-03, -1.3452e-01, -1.1709e-01, -8.5242e-02],
        [ 4.3661e-02,  3.3325e-02, -7.7510e-02,  8.1618e-02, -7.8005e-02,
         -1.3573e-01, -2.5096e-02, -3.2416e-02,  1.3646e-01, -1.3961e-01,
         -2.7059e-02, -1.2513e-01,  6.2434e-02,  1.0024e-01, -9.9094e-02,
         -7.5195e-02,  7.9039e-02,  1.1273e-01,  1.2019e-01,  6.7251e-02,
         -1.3087e-01,  3.0930e-02, -8.0486e-02,  1.4882e-01,  5.4236e-03,
         -6.1695e-02,  6.0822e-02,  9.7030e-02, -3.0092e-02, -1.0603e-01,
         -1.2786e-01, -4.9689e-03, -1.2311e-02,  7.6463e-02, -1.4557e-01,
          5.9748e-03, -1.0503e-01,  1.4319e-01, -1.0756e-01,  5.3949e-02]],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.linear_net.3.bias
tensor([ 0.1648,  0.0744, -0.0017, -0.0909, -0.1308, -0.0109,  0.1197, -0.1440,
         0.0546,  0.0123,  0.0982, -0.0400, -0.1743,  0.0116, -0.0247,  0.1516,
         0.0360, -0.1574, -0.1504,  0.1289], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm1.weight
tensor([0.9396, 0.9300, 0.9300, 0.9381, 0.9682, 0.9456, 0.9689, 0.9987, 1.0751,
        1.0605, 1.0138, 1.0163, 1.0365, 1.0374, 0.9790, 1.0330, 1.0365, 1.0252,
        1.0579, 1.0155], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm1.bias
tensor([ 0.0163,  0.0073,  0.0190, -0.0112, -0.0197, -0.0222,  0.0149, -0.0190,
        -0.0116, -0.0043, -0.0115,  0.0034, -0.0272,  0.0222,  0.0273,  0.0204,
         0.0034, -0.0052, -0.0045,  0.0025], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm2.weight
tensor([0.9358, 0.9437, 0.9390, 0.9421, 0.9751, 0.9474, 0.9740, 1.0095, 1.0527,
        1.0619, 1.0176, 1.0164, 1.0390, 1.0309, 0.9834, 1.0287, 1.0256, 1.0246,
        1.0119, 1.0209], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm2.bias
tensor([ 0.0156,  0.0139,  0.0103, -0.0061, -0.0255, -0.0212,  0.0101, -0.0075,
        -0.0196, -0.0013, -0.0125,  0.0086, -0.0288,  0.0186,  0.0214,  0.0181,
         0.0008,  0.0014, -0.0007,  0.0107], device='cuda:0')
--------------------------------------------------------------------------------
output_net.0.weight
tensor([[-0.0816,  0.0147, -0.1951,  0.1983, -0.0452,  0.1409, -0.0768,  0.0565,
         -0.0117, -0.1893, -0.1837, -0.1965,  0.2024, -0.2540,  0.1489,  0.0009,
          0.1001,  0.1820,  0.1264,  0.1042],
        [-0.1496,  0.0158, -0.0341,  0.0949, -0.0672,  0.0245, -0.0982, -0.0291,
         -0.1021, -0.1133, -0.0657, -0.0361, -0.1504, -0.1184, -0.0008,  0.1257,
```

```
 -0.0609,  0.0737, -0.1615, -0.0200],
[ 0.0108,  0.1667, -0.1838, -0.1060,  0.1212, -0.1927,  0.0775,  0.1950,
 -0.0243,  0.1296,  0.0480,  0.1799, -0.1877, -0.2204,  0.1795,  0.0848,
 -0.1094,  0.0009,  0.0696, -0.1727],
[-0.0634, -0.0991, -0.0775, -0.1497, -0.1018,  0.1288,  0.0914,  0.0043,
 -0.0511,  0.1884, -0.0576,  0.1532, -0.1153, -0.1330, -0.1605,  0.0673,
 -0.1539,  0.2159,  0.0527, -0.2386],
[-0.1589,  0.0390,  0.1336, -0.0975, -0.1481,  0.1178, -0.1209,  0.1676,
  0.1246,  0.0674, -0.0075,  0.1742,  0.0945, -0.0320, -0.0982, -0.1244,
  0.1809, -0.0727, -0.1653,  0.0008],
[ 0.0338, -0.1795,  0.0676,  0.0323,  0.1207, -0.0214,  0.1670,  0.1771,
 -0.1693, -0.0377,  0.1707,  0.0935, -0.1036,  0.2048,  0.0354, -0.0684,
  0.1122,  0.0865, -0.1231,  0.1799],
[-0.1011, -0.0876, -0.0909, -0.1104,  0.0821, -0.0018, -0.0774, -0.0451,
 -0.1021,  0.1332, -0.0855, -0.1273, -0.0593,  0.0882, -0.0219,  0.0701,
 -0.1187,  0.0775,  0.0406,  0.0539],
[-0.1309,  0.0795, -0.1157,  0.1148,  0.0980, -0.1643,  0.0285,  0.0188,
 -0.2526,  0.2671, -0.2098, -0.0447, -0.2249, -0.1109,  0.0846,  0.0896,
 -0.0183,  0.1017,  0.0394, -0.0567],
[ 0.0912, -0.1937, -0.0218, -0.1060,  0.2054, -0.0417,  0.1664, -0.2029,
 -0.0647,  0.1327,  0.2332,  0.1199,  0.0748, -0.0951, -0.0344, -0.0279,
 -0.0283,  0.0078, -0.1438,  0.1943],
[-0.0670, -0.0496,  0.0806,  0.1742, -0.1293,  0.0829,  0.1996,  0.1581,
 -0.1119, -0.0090, -0.1220,  0.1587, -0.1923, -0.1196,  0.1142, -0.0219,
 -0.0827, -0.0959, -0.0864,  0.1874],
[-0.1499, -0.0712, -0.1355, -0.0457,  0.0589, -0.0319, -0.1488, -0.1469,
 -0.0398, -0.0342, -0.0393, -0.0269, -0.0269,  0.1031,  0.0007, -0.1315,
 -0.0559,  0.1993,  0.0150,  0.1724],
[ 0.0102,  0.1204,  0.0441, -0.1734, -0.0578, -0.0809, -0.0260, -0.0595,
 -0.0059,  0.0627, -0.1455,  0.2186, -0.2483, -0.1481,  0.0600,  0.0240,
 -0.1805, -0.0649,  0.1971, -0.1495],
[ 0.1032, -0.1893,  0.1720,  0.0608,  0.0286,  0.1194, -0.0310,  0.1692,
  0.1535, -0.2168, -0.1940, -0.0538, -0.1416,  0.0813, -0.0904,  0.1175,
 -0.0489,  0.0770, -0.0663,  0.0962],
[-0.0926, -0.1377,  0.0879,  0.1188,  0.1779,  0.0755, -0.0087,  0.2017,
 -0.0116, -0.1479, -0.1904,  0.2018,  0.1914,  0.1316, -0.1230,  0.0143,
  0.0769,  0.1075, -0.1804,  0.0474],
[-0.1962, -0.1129, -0.0937, -0.0673, -0.0447, -0.1448,  0.1407, -0.0119,
  0.0514,  0.0048, -0.1765, -0.2022,  0.0758,  0.0934,  0.0443,  0.2237,
  0.1992, -0.2136, -0.1207, -0.0744],
[-0.1599,  0.0442,  0.1839,  0.0757, -0.1022,  0.0786,  0.0915,  0.0813,
 -0.0696, -0.0663,  0.1158, -0.2448,  0.2483, -0.2298, -0.1042, -0.0984,
 -0.0203, -0.1375, -0.0726, -0.0660],
[ 0.0139, -0.0754,  0.0025, -0.0050,  0.0846, -0.0111,  0.2290, -0.0266,
 -0.1356,  0.2801,  0.0338,  0.1244,  0.0690,  0.0274,  0.1202,  0.1166,
 -0.2252,  0.1212,  0.1231, -0.1060],
[ 0.0201, -0.1480, -0.1131,  0.0767,  0.1416,  0.1880,  0.0530,  0.1244,
  0.1275,  0.1383, -0.1406, -0.2292,  0.1885, -0.1184,  0.0593, -0.1772,
```

```
          0.0010, -0.1669,  0.0737, -0.0128],
        [ 0.1982, -0.2447,  0.0014, -0.1655, -0.2016, -0.0587,  0.2072,  0.1526,
          0.1168,  0.0668, -0.0523, -0.0633, -0.0236,  0.1005,  0.1350, -0.1080,
          0.0208, -0.1781, -0.0862, -0.0426],
        [-0.0486, -0.1846, -0.0814,  0.1086,  0.1469, -0.1948, -0.2525, -0.1108,
         -0.0027,  0.0156,  0.2334,  0.1658, -0.1926, -0.0633,  0.1433, -0.0779,
          0.1051,  0.0123,  0.1028,  0.0751]], device='cuda:0')
--------------------------------------------------------------------------------
output_net.0.bias
tensor([ 0.1584,  0.1770, -0.0362,  0.1642,  0.1733, -0.0063, -0.1995, -0.0186,
        -0.1236, -0.0538,  0.0848, -0.1379, -0.1080,  0.1475,  0.1875, -0.1249,
         0.0336, -0.0931,  0.0079,  0.0063], device='cuda:0')
--------------------------------------------------------------------------------
output_net.1.weight
tensor([0.9960, 0.9426, 0.9671, 0.9837, 0.9829, 0.9675, 0.9716, 0.9605, 0.9859,
        0.9237, 0.9679, 0.9296, 0.9697, 0.9835, 0.9627, 1.0000, 0.9683, 1.0000,
        0.9431, 0.9830], device='cuda:0')
--------------------------------------------------------------------------------
output_net.1.bias
tensor([-0.0037, -0.0333, -0.0382, -0.0256, -0.0280, -0.0379, -0.0287, -0.0307,
        -0.0248, -0.0345, -0.0255, -0.0483, -0.0342, -0.0232, -0.0396,  0.0000,
        -0.0316,  0.0000, -0.0448, -0.0275], device='cuda:0')
--------------------------------------------------------------------------------
output_net.4.weight
tensor([[ 0.0366, -0.1320,  0.1119,  …, -0.1406,  0.0901, -0.1988],
        [ 0.1134, -0.0612, -0.1936,  …, -0.1556, -0.2561, -0.0085],
        [-0.1112, -0.2155, -0.0735,  …, -0.2228, -0.0344,  0.1170],
        …,
        [ 0.1471,  0.0547,  0.0560,  …,  0.1426, -0.1621, -0.0674],
        [ 0.1501,  0.0919, -0.0303,  …, -0.1696, -0.1823, -0.0469],
        [-0.0997,  0.0882,  0.2057,  …,  0.0800,  0.2577,  0.2096]],
       device='cuda:0')
--------------------------------------------------------------------------------
output_net.4.bias
tensor([-0.0599,  0.1118,  0.0322, -0.1698,  0.1371, -0.0069,  0.1431, -0.0647,
         0.0739,  0.1638, -0.0032,  0.1157, -0.1531, -0.1136, -0.0865,  0.1462,
         0.1808,  0.1902, -0.1935,  0.0231,  0.2107, -0.0762, -0.0406, -0.0979,
         0.2018,  0.0099,  0.0674, -0.1460,  0.2039, -0.0642,  0.1791, -0.1101,
        -0.0759, -0.2191,  0.0057,  0.1482,  0.1093, -0.1962,  0.1406, -0.0937,
        -0.0300,  0.1978, -0.1120, -0.1773,  0.0014,  0.0028,  0.1969,  0.0286,
         0.0676,  0.0159, -0.1298,  0.0333,  0.0094,  0.1506,  0.0276, -0.0534,
        -0.0827, -0.0550,  0.0434,  0.0928, -0.0418, -0.1334, -0.2495,  0.0787,
        -0.1908, -0.0075, -0.0171,  0.0591,  0.1381, -0.1255,  0.1527,  0.2297,
        -0.1463, -0.0008, -0.0174, -0.0580, -0.1726,  0.0482, -0.1520, -0.0973,
        -0.1120,  0.0351,  0.0038,  0.1275, -0.0196, -0.0968,  0.0426, -0.2152,
         0.0651, -0.1482, -0.1684, -0.1656,  0.1080,  0.0782,  0.1052, -0.2068,
         0.0782,  0.1616,  0.0409, -0.1633], device='cuda:0')
--------------------------------------------------------------------------------
```

```
[ ]:
```