model_131_beta=5_using_the_alter_algo (1)

September 26, 2023

[1]:

```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)
```

```python
# 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 * 5 #/ (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)
```

```python
        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)
```

```python
        # 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)
```

```python
        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

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

```python
        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):
        self.warmup = warmup
        self.max_num_iters = max_iters
```

```python
        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])
```

[2]:

```
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
```

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

```
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger folder:
saved_models/tutorial6/ReverseTask/lightning_logs
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)
```

```
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]
```

[3]:

```python
# @title # model_131_beta=5_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:  96.43%
Test accuracy: 96.44%
```

[6]:

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

[7]:

```python
#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
```

[8]:
```
for odict in w:
    print(odict)
    print(w[str(odict)])
    #print(str( (w[str(odict)][5])))
    print("-------------------------------------------------------------------------------")
```

```
input_net.1.weight
tensor([[-0.0350,  0.0685, -0.0798,  …,  0.0073,  0.0280, -0.0734],
        [-0.0593, -0.0045, -0.0622,  …,  0.0790, -0.0287,  0.0446],
        [-0.0612, -0.0769,  0.0088,  …,  0.0126,  0.1124, -0.0317],
        …,
        [-0.0486, -0.0088,  0.0556,  …,  0.0201, -0.0175,  0.0331],
        [ 0.0679,  0.1667,  0.0706,  …, -0.0746,  0.0204, -0.0261],
        [-0.0220, -0.0633, -0.0132,  …, -0.0077,  0.0529,  0.0316]],
       device='cuda:0')
-------------------------------------------------------------------------------
input_net.1.bias
tensor([-0.0078, -0.0101, -0.0319, -0.0678, -0.0408, -0.0384, -0.0641, -0.0252,
         0.0296,  0.0166, -0.0713, -0.1129,  0.0168, -0.0400,  0.0400, -0.1022,
         0.0672, -0.0757,  0.1145, -0.0316], device='cuda:0')
-------------------------------------------------------------------------------
transformer.layers.0.self_attn.qkv_proj.weight
tensor([[-0.3153, -0.2766, -0.1042,  …,  0.2538, -0.0402, -0.1034],
        [ 0.4574, -0.4184, -0.0770,  …, -0.1444,  0.1758, -0.1419],
        [-0.2668,  0.1546, -0.2720,  …, -0.1659,  0.0114, -0.1847],
        …,
        [ 0.0966,  0.0813, -0.2479,  …,  0.1152,  0.1799, -0.1152],
        [ 0.1382,  0.0292, -0.0520,  …, -0.2155,  0.3142,  0.0095],
```

```
        [ 0.1878,  0.0312, -0.2110,  …, -0.2603,  0.4794,  0.0636]],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.qkv_proj.bias
tensor([-1.7811e-02,  2.9813e-03, -2.3117e-02,  2.2288e-03, -1.7551e-02,
         2.6426e-02,  7.9288e-03, -1.1374e-02,  1.1349e-02, -3.3630e-02,
         3.2969e-02,  3.9939e-02,  1.1011e-02, -3.2553e-02,  1.7364e-04,
         1.7726e-02, -4.5852e-02,  1.1887e-02,  3.2100e-02, -1.5062e-02,
         4.8714e-03,  5.6186e-03,  9.0235e-04, -5.1274e-03, -3.3010e-03,
        -7.7485e-04, -1.5904e-03, -2.1712e-03,  4.9868e-03,  8.7120e-05,
         4.5721e-03, -2.8554e-03,  2.6499e-04,  1.0180e-03, -1.2425e-03,
         3.6793e-03, -2.4732e-03, -5.6559e-03,  8.2898e-04, -1.9432e-03,
         3.2447e-03,  3.9199e-02,  2.3096e-03,  6.8915e-03, -1.0076e-02,
        -1.8254e-02,  8.0812e-03,  2.6624e-02,  2.1791e-03,  2.7012e-02,
        -2.6243e-03, -1.0607e-02, -3.0037e-02, -4.0932e-03,  4.5547e-03,
         9.9671e-03,  1.4284e-02, -2.5580e-03,  2.6401e-02,  9.8760e-03],
       device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.o_proj.weight
tensor([[-0.0561,  0.1635, -0.0807,  0.2579, -0.0500, -0.2085, -0.0503,  0.5447,
          0.2494, -0.1249,  0.0932,  0.0022, -0.1440,  0.0008, -0.0540, -0.1055,
          0.2085,  0.4143,  0.3724, -0.0568],
        [ 0.3465, -0.2444,  0.1998, -0.4349,  0.1407, -0.2882,  0.0649, -0.0542,
          0.4083,  0.1792,  0.4128,  0.1486,  0.1218, -0.4548, -0.2376,  0.2202,
          0.1111, -0.1140,  0.3099, -0.1224],
        [-0.1309,  0.4016,  0.1954, -0.1382, -0.4477, -0.0917,  0.0230,  0.4567,
          0.1371,  0.1725,  0.1224, -0.0262,  0.0775,  0.1413,  0.2182, -0.2172,
          0.0357, -0.3907, -0.2641, -0.0980],
        [-0.2364, -0.2379, -0.2931,  0.2924, -0.3540, -0.2896,  0.3280, -0.2803,
         -0.0043,  0.3162,  0.0186,  0.0011, -0.4576, -0.2048,  0.2672, -0.1129,
         -0.0891,  0.3358,  0.2540, -0.2462],
        [-0.4033, -0.1171, -0.2037,  0.1873,  0.1364,  0.0063,  0.2459,  0.0925,
          0.4967,  0.1531, -0.3072,  0.1530,  0.2339,  0.5673,  0.2900, -0.4863,
         -0.0632,  0.1275,  0.1382, -0.3035],
        [ 0.0329,  0.0058,  0.0156,  0.0251,  0.1942, -0.2228,  0.0251,  0.1073,
          0.0176, -0.1682, -0.0877, -0.2098, -0.0178, -0.0184, -0.3567,  0.0540,
          0.3347,  0.0841,  0.3023,  0.0996],
        [-0.1856, -0.1732,  0.2754,  0.3950,  0.2693,  0.0880,  0.3187,  0.4246,
         -0.2909,  0.3398,  0.0420, -0.2680,  0.2598, -0.4601, -0.0167, -0.0125,
          0.2852,  0.2653, -0.0483,  0.0115],
        [ 0.2836, -0.3982, -0.2808,  0.4197,  0.2101, -0.3014, -0.1938, -0.3516,
         -0.4087, -0.1660,  0.1306,  0.1787,  0.1108, -0.0202,  0.4513,  0.2244,
         -0.4627, -0.1655,  0.1432, -0.2849],
        [ 0.2850,  0.1114,  0.0448, -0.1149,  0.1418, -0.2075,  0.0031,  0.5527,
         -0.4517,  0.0035, -0.2976,  0.0841, -0.4422, -0.1947, -0.0812,  0.0541,
          0.2471, -0.1702, -0.3041, -0.2839],
        [-0.2269, -0.1025, -0.0519, -0.0754,  0.3419,  0.3692, -0.1761,  0.2324,
          0.1175, -0.0496, -0.2150, -0.4080,  0.3742,  0.0311,  0.1823, -0.3418,
```

```
          0.3768, -0.2521,  0.0551,  0.1295],
        [-0.0824, -0.3372,  0.3142,  0.1666, -0.3604,  0.1779, -0.2162,  0.0792,
         -0.1312, -0.3694, -0.1970,  0.2463,  0.3592, -0.0261,  0.0716, -0.2309,
         -0.4591,  0.0559, -0.4054, -0.3037],
        [ 0.0946, -0.2909, -0.0484, -0.1087,  0.2746, -0.1404,  0.1074, -0.3358,
         -0.0763, -0.0179,  0.2052, -0.2443, -0.0406, -0.1333, -0.3785, -0.2004,
          0.2687,  0.1204, -0.4764, -0.4248],
        [ 0.0488, -0.0838, -0.3672,  0.2643, -0.0458, -0.1984, -0.0770,  0.3358,
         -0.2281,  0.2786,  0.0602, -0.2121, -0.2063,  0.4532,  0.3143, -0.1108,
          0.3976,  0.1375, -0.1340,  0.2178],
        [-0.2663,  0.3049, -0.4416,  0.3625, -0.1195,  0.4653,  0.2700, -0.4942,
         -0.0173, -0.0822, -0.1941, -0.2445, -0.0746, -0.3245,  0.1120,  0.4829,
          0.2310, -0.4397, -0.2442,  0.3163],
        [ 0.3408, -0.3663,  0.2205, -0.1195, -0.2447, -0.0662,  0.2423, -0.0904,
         -0.2707,  0.0135, -0.0933,  0.0013, -0.4447, -0.0634,  0.4043,  0.2842,
          0.1576, -0.3088,  0.1881,  0.4370],
        [ 0.4147,  0.0044,  0.0421, -0.0540,  0.1090,  0.2088,  0.2692, -0.1285,
         -0.0620,  0.5047,  0.3181, -0.0635,  0.1228,  0.0093,  0.0655,  0.0918,
          0.2079,  0.2618, -0.4164,  0.2862],
        [-0.2366, -0.2987, -0.3659, -0.2327, -0.2841, -0.2273,  0.3959, -0.0363,
          0.1286, -0.4077, -0.4396,  0.3960, -0.1485, -0.0665,  0.4143,  0.2469,
         -0.1992,  0.3175,  0.0671,  0.0961],
        [-0.1343,  0.3321,  0.1444, -0.4329, -0.0902,  0.1879, -0.3371, -0.2439,
         -0.3697, -0.4727,  0.2368, -0.2086, -0.3258, -0.1738, -0.3737, -0.1899,
          0.2266,  0.5397,  0.2059,  0.1296],
        [ 0.3271,  0.0539,  0.0684, -0.4994,  0.0832, -0.0013,  0.1694, -0.0109,
          0.4323,  0.2851, -0.4086,  0.0275, -0.2293,  0.4443, -0.2778,  0.5211,
         -0.0794, -0.1646, -0.2879, -0.1217],
        [ 0.0626, -0.0186, -0.3938, -0.0814,  0.1202, -0.4104, -0.2289, -0.1944,
          0.1956,  0.2898, -0.3643, -0.2032,  0.4103,  0.2483,  0.0933,  0.0332,
         -0.0693, -0.5280,  0.3900, -0.3267]], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.self_attn.o_proj.bias
tensor([ 0.0227,  0.0008,  0.0470,  0.0096, -0.0278, -0.0054,  0.0223, -0.0069,
         0.0179, -0.0026, -0.0281, -0.0433, -0.0075,  0.0051, -0.0285,  0.0028,
        -0.0051, -0.0127,  0.0116,  0.0135], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.linear_net.0.weight
tensor([[ 1.0081e-01, -1.1280e-01, -6.0030e-02, -1.2335e-01,  2.6579e-01,
         -1.8753e-02,  2.1083e-02,  6.8688e-02,  1.7164e-01,  8.3895e-02,
         -1.5348e-01, -6.9548e-02, -2.0680e-02,  2.0667e-01,  1.3938e-02,
          8.2449e-02, -2.2301e-01, -1.3742e-01, -9.0042e-03,  2.4597e-01],
        [ 1.8066e-01, -1.9904e-01,  2.7191e-02,  2.0252e-02,  1.0874e-01,
          1.0736e-01, -2.3455e-01, -3.3244e-01,  1.9550e-01, -1.7978e-01,
         -5.8422e-02, -2.5888e-01, -1.8648e-01, -1.1687e-01, -2.2839e-01,
          5.7569e-02, -1.3592e-01,  7.4454e-02, -1.8738e-02, -1.6667e-01],
        [ 2.9056e-01, -2.8142e-01, -4.7826e-02, -8.6005e-02,  1.1009e-01,
         -1.4801e-01,  8.4692e-02,  7.3059e-03, -1.1133e-01,  1.7457e-01,
```

```
 -2.8460e-02,  1.5316e-01, -1.0700e-01, -1.4866e-01,  9.9235e-02,
  1.4024e-01, -2.7735e-01,  1.0441e-01,  3.5674e-02, -2.0927e-01],
[-1.1408e-01,  1.4796e-01,  7.6205e-02, -1.9402e-02,  1.8985e-01,
 -1.6008e-01,  2.1962e-01, -5.6611e-02,  2.3029e-02,  9.3446e-02,
 -8.5535e-02,  5.8830e-02,  3.9267e-02,  1.3497e-02, -8.4922e-02,
  1.8915e-01,  1.1432e-01,  1.5396e-01, -8.1322e-02,  6.5433e-03],
[ 2.1375e-01, -3.4746e-02,  1.3161e-01, -2.6675e-01, -6.5622e-02,
  7.2379e-02,  1.4443e-01,  2.3490e-01,  8.4233e-02, -1.7765e-01,
  2.8794e-02, -1.8392e-01, -7.0213e-02,  2.2752e-01, -1.7347e-01,
  1.5594e-02,  1.0736e-01, -6.6231e-02, -7.8538e-02,  1.5385e-01],
[ 5.3847e-02, -3.0590e-02,  2.8912e-01,  2.4748e-02,  2.5872e-01,
  7.7096e-02, -6.2238e-02, -1.9296e-01, -1.4864e-01,  2.1972e-01,
 -1.5257e-01, -1.3061e-01, -2.2452e-01, -3.6663e-02,  1.2298e-03,
 -8.8945e-02,  1.2776e-01,  1.2797e-01,  1.5573e-01, -3.6803e-02],
[ 1.3682e-01,  4.3537e-02,  1.6994e-01, -1.1604e-01, -8.2807e-02,
 -1.8346e-01, -1.9698e-01, -2.4905e-01,  2.4492e-01, -7.8375e-02,
  7.8261e-02,  2.0699e-01,  1.0408e-01,  6.2221e-03,  3.8247e-02,
  2.3763e-02, -1.6221e-01,  1.6194e-01,  5.3740e-02,  5.5058e-02],
[-3.8699e-03, -1.6406e-01, -3.3786e-02,  6.9038e-02,  7.5954e-02,
 -4.8254e-02, -3.5626e-02,  3.2688e-01,  5.6288e-02, -8.7590e-02,
  6.9149e-02,  1.6809e-01,  1.1452e-03,  2.2325e-02, -8.9165e-02,
  8.0754e-02,  3.7438e-01, -9.1631e-02, -1.3601e-01,  6.1784e-02],
[-1.0020e-01,  1.5688e-01,  6.3029e-02,  1.3674e-01,  1.5696e-01,
 -7.6250e-02,  1.1726e-02, -7.4003e-02,  1.2450e-01,  1.7308e-01,
 -1.7690e-01,  2.3575e-01, -1.4707e-01, -2.9178e-02, -1.8833e-02,
  1.1499e-02,  2.3075e-01, -1.4018e-01, -1.3028e-02,  1.0793e-01],
[-8.8627e-02, -2.2220e-01,  2.6392e-01,  1.0811e-01, -1.6369e-01,
 -1.6640e-01,  2.6554e-01, -8.1753e-02,  1.0384e-01,  2.4076e-01,
 -1.1924e-01, -4.3617e-02,  1.6663e-01,  1.4194e-01, -6.1198e-02,
 -5.9288e-02,  4.6782e-02,  1.2337e-01, -1.6470e-01, -4.3966e-02],
[ 1.5489e-01, -1.8310e-01, -2.8319e-01,  4.2011e-02, -7.8555e-02,
 -1.4948e-02, -7.4731e-02,  4.7588e-02, -7.2739e-02,  4.7656e-02,
  2.0121e-01, -7.1954e-02, -2.0366e-01, -1.9584e-01, -1.9727e-01,
  1.6791e-01, -1.5247e-01,  9.6656e-03, -6.2167e-02,  8.4978e-02],
[ 4.1129e-02, -1.0137e-01,  1.8893e-01,  1.4836e-01,  1.6117e-01,
  5.9964e-03, -4.2136e-02,  9.5951e-03,  1.0539e-01, -1.5031e-01,
  1.4680e-01, -2.6335e-01,  9.5209e-02, -2.8765e-02, -1.4334e-01,
  1.9738e-01, -2.1301e-01,  7.3202e-02, -2.1626e-01,  8.8611e-03],
[-2.4364e-02, -1.5860e-01, -2.8256e-01, -2.8323e-01,  1.1490e-01,
 -4.0531e-02, -2.4696e-01,  2.7062e-03, -8.5555e-02, -1.7638e-01,
 -2.2589e-01,  2.4887e-01, -1.0585e-01,  2.5369e-01,  1.7694e-01,
 -1.1018e-01,  9.8831e-02,  1.1467e-01,  2.6014e-01,  2.2450e-01],
[ 7.9117e-02,  2.4783e-03,  2.0816e-01, -2.3550e-01, -2.0341e-01,
 -4.7891e-02, -1.8299e-01,  1.0443e-01,  1.5370e-01, -1.1896e-01,
 -1.8815e-01, -1.0029e-01,  1.1515e-02,  8.8213e-03, -1.9885e-01,
 -1.6637e-01,  3.3129e-02,  3.6321e-02, -9.4554e-02,  2.4880e-01],
[-3.5608e-02,  2.8368e-02,  2.2389e-02,  1.5702e-01,  9.5702e-02,
 -2.6935e-01, -1.7784e-01,  2.0253e-01,  1.3493e-01, -3.4035e-01,
```

```
 -2.5647e-01, -1.5836e-01,  6.5121e-02,  4.5307e-02, -6.9689e-02,
 -1.0256e-01,  1.8219e-01, -1.5555e-01,  2.1186e-01,  9.2950e-02],
[-1.1007e-01,  2.0456e-01,  1.4453e-01, -2.1726e-01, -1.1075e-01,
 -1.8246e-02, -1.2543e-01,  1.0402e-01, -5.0625e-02, -4.9634e-02,
  7.9730e-02, -8.5710e-02, -1.0659e-01, -2.2609e-01, -1.4576e-02,
  1.7813e-04,  7.7852e-05,  1.6353e-01, -2.2159e-01, -8.3213e-02],
[-1.6652e-01, -1.5665e-01, -1.6065e-01, -3.6507e-01, -6.8333e-02,
 -6.1109e-02,  8.0528e-02,  1.2229e-01, -2.7050e-02,  6.1443e-02,
  1.6628e-01, -1.3018e-01, -9.1585e-02,  6.0006e-02, -1.5650e-02,
 -8.5284e-02, -6.3740e-02, -6.6584e-04,  2.2884e-02,  1.3897e-01],
[ 2.2398e-01,  4.7928e-03,  1.8235e-01, -1.8345e-01, -8.2031e-02,
  1.1853e-01,  9.7150e-02, -2.1332e-01, -2.1106e-01, -5.0429e-02,
  2.4981e-02, -8.5940e-02, -1.4319e-01,  2.3121e-01, -2.4430e-01,
  1.7826e-01, -9.5972e-02,  6.7092e-03,  1.4288e-01, -5.8270e-02],
[-5.2723e-02, -1.7288e-01,  1.4121e-01, -2.5156e-01, -2.4295e-01,
 -1.4722e-01,  4.2814e-02,  2.6569e-01,  2.1916e-01,  4.7999e-03,
  7.5900e-02,  1.4700e-01, -6.5959e-02,  1.1118e-02,  6.9930e-02,
  2.4275e-01, -1.2800e-01,  8.9289e-03, -5.8206e-02, -1.1403e-01],
[-7.0400e-02, -2.0967e-01,  4.3083e-02, -2.8345e-01,  7.6326e-02,
  1.8434e-01,  1.6289e-01,  6.4822e-02,  1.3406e-02, -1.4717e-01,
 -1.1085e-01, -5.2311e-02,  2.4810e-01,  2.8571e-01, -1.4927e-01,
  2.1307e-02, -8.7841e-02,  5.5981e-02,  4.1507e-02,  1.2353e-01],
[-1.4867e-01,  1.1393e-01,  1.1143e-01,  9.6779e-02,  5.1350e-02,
 -1.7810e-01, -2.0042e-01, -3.2483e-02, -3.1102e-03,  5.9949e-02,
 -7.3963e-02, -1.2614e-01, -8.4130e-02,  1.8069e-01, -1.6717e-01,
 -2.6251e-02,  2.8710e-01, -2.4756e-02, -1.7217e-01, -1.4203e-01],
[-1.9717e-02, -2.2141e-01, -1.5893e-01,  2.7474e-01,  2.2768e-01,
 -1.0554e-02,  2.9007e-01, -1.3573e-01,  2.9793e-01, -5.2626e-02,
 -1.5448e-01,  5.0861e-02,  1.3969e-01,  5.5526e-03,  1.4462e-01,
  1.0523e-01, -2.3963e-02,  1.0667e-01, -9.3744e-02, -3.9539e-02],
[ 9.1296e-02,  1.2502e-01, -1.8169e-01, -8.3498e-02, -8.4436e-02,
  2.1079e-01,  3.5730e-02, -2.8622e-04, -4.5666e-02, -8.0783e-02,
  9.4029e-02, -7.3066e-02,  2.2090e-01,  2.5814e-01, -1.5885e-01,
  1.6672e-01,  3.6256e-02,  5.5467e-02, -1.4486e-01,  3.3215e-01],
[-1.9460e-01,  7.7984e-03, -3.6283e-02, -3.9857e-03, -8.9651e-03,
 -1.9237e-01,  9.1528e-02, -5.8096e-02,  9.4595e-02,  1.0189e-01,
 -1.8979e-01,  1.7556e-01, -1.3056e-01,  6.6502e-02, -1.1512e-01,
 -1.7887e-01, -1.1243e-01, -6.3557e-02, -1.5252e-01,  2.2268e-01],
[ 2.5179e-02, -2.0424e-02,  1.8109e-01,  5.2849e-03, -6.6637e-02,
 -5.0766e-02, -2.2413e-01, -2.0471e-02, -1.1068e-02,  1.9203e-01,
 -1.0428e-01,  8.2092e-02, -2.0403e-01,  2.3655e-02, -9.2846e-02,
  1.4294e-01, -1.2462e-02,  1.6991e-02,  2.6454e-01, -9.8419e-02],
[-1.0658e-01, -3.0663e-02,  1.1518e-01, -5.3677e-02, -1.4095e-02,
 -1.7696e-01,  5.5555e-02, -1.1526e-01, -3.2333e-01, -1.3216e-01,
 -7.4639e-02, -7.4365e-02, -1.2686e-01, -1.0022e-01,  1.5824e-01,
  7.1655e-02,  5.7034e-02, -8.8555e-02,  2.0278e-02,  2.3225e-01],
[ 1.1556e-01,  1.8376e-01, -1.3654e-01,  5.5838e-02, -1.1504e-01,
  2.0594e-01,  2.4960e-01, -2.2187e-01,  1.2761e-01, -7.6047e-02,
```

```
 2.1139e-01, -7.3514e-02,  1.0516e-01,  8.4679e-02,  1.1199e-01,
-1.0717e-01,  2.3421e-01,  1.4425e-01, -7.9300e-02, -1.3216e-01],
[-2.8392e-01,  6.6125e-02,  2.2224e-01,  1.0788e-02, -3.6848e-02,
 8.1014e-02,  1.8703e-01, -4.1807e-02,  1.1698e-01, -2.0357e-01,
 4.9649e-02, -1.2345e-01,  7.1768e-02,  2.3302e-01, -2.2514e-01,
 1.0312e-01,  1.1526e-01,  1.2616e-01, -4.2069e-02, -1.9979e-01],
[ 3.2127e-02,  6.9470e-02, -1.9818e-01, -5.8665e-02, -1.2767e-01,
 1.3541e-01,  9.0707e-02, -7.6556e-02, -5.3476e-02,  3.2450e-01,
-2.3710e-01,  1.0640e-01,  4.9404e-02,  1.4958e-01, -1.8940e-01,
 2.7082e-01, -1.4616e-01,  3.4328e-02, -1.1161e-01,  1.6754e-01],
[-2.2004e-01, -1.3485e-01,  3.4348e-01,  1.9106e-01, -1.7818e-01,
-4.6676e-02,  1.9820e-01, -9.3993e-02, -1.0819e-01,  1.8993e-01,
-1.9441e-01, -9.5396e-02,  3.8087e-02, -2.8353e-02, -2.8678e-01,
 6.9345e-03,  1.2209e-01,  7.6052e-02, -2.7591e-02,  2.1652e-01],
[-6.8371e-02,  7.3124e-02, -2.2918e-01,  5.5559e-02,  2.7769e-02,
 2.9503e-02,  1.4948e-01,  3.4118e-02,  5.9607e-02, -1.2044e-01,
 1.4172e-01,  1.5949e-01,  4.0694e-03,  9.6018e-02,  4.3317e-02,
 5.3628e-02, -2.4995e-01,  4.2263e-02,  3.3753e-02,  2.5133e-01],
[ 1.9841e-01, -1.9953e-01,  2.0020e-01,  1.2549e-02,  1.5633e-01,
 8.3597e-02,  8.3395e-02,  4.2114e-02, -2.6468e-01,  6.1801e-02,
-2.5628e-01, -2.3570e-01, -2.2244e-01,  1.5146e-01,  1.0537e-03,
-1.5893e-01, -2.2630e-01,  1.1880e-01,  1.3846e-01,  1.5683e-01],
[-2.2040e-01, -1.5083e-01,  1.4260e-01,  1.3653e-01,  4.8438e-02,
-2.3531e-01,  1.6014e-01, -2.2250e-01, -9.2855e-02, -1.2444e-01,
-7.2092e-02, -1.2280e-01, -8.5745e-02,  2.4033e-02, -1.6683e-01,
 1.4762e-01, -1.3409e-01,  1.2922e-01, -7.6339e-02, -1.7888e-01],
[-1.1401e-01,  5.2871e-02,  2.5606e-01,  6.0470e-03,  1.9133e-02,
 2.1043e-03, -1.9241e-01,  8.6292e-02,  9.4305e-02,  1.1501e-01,
 7.8471e-02, -1.0652e-01, -1.3367e-01,  1.1092e-01, -5.5601e-03,
-2.2788e-01,  1.5895e-01, -1.2844e-01, -2.6486e-02, -1.8550e-01],
[-1.4241e-01, -2.0878e-01,  1.1092e-01,  8.5159e-02,  6.6114e-02,
-7.4432e-02,  1.8037e-01, -1.9222e-01,  2.3568e-01,  8.9741e-02,
 7.4712e-03,  8.5292e-03, -1.7894e-01, -1.1879e-01, -1.9464e-01,
-1.1769e-01, -1.6080e-01, -5.0021e-02, -3.2172e-02,  1.0304e-01],
[ 4.6436e-02, -2.1666e-01,  1.8461e-01,  5.4627e-02, -1.0288e-01,
 1.1751e-01,  1.7380e-01, -8.5338e-02,  7.5232e-03, -7.8040e-02,
-5.6229e-03,  1.3254e-02, -1.2529e-01, -5.4797e-03, -1.3756e-03,
 1.0750e-01,  1.5143e-03, -8.5918e-02, -2.3492e-01,  9.4934e-02],
[-2.2339e-01, -1.3843e-01, -1.7333e-01,  2.8268e-01,  1.5902e-01,
-2.8640e-01,  1.1244e-01, -2.5112e-01,  8.5710e-02,  2.5459e-01,
-1.3391e-01,  4.8664e-02,  5.0090e-02,  1.0176e-01,  8.6884e-02,
 2.5951e-01,  4.7347e-02, -4.1863e-02, -2.1704e-01, -6.0255e-02],
[ 9.5682e-02,  2.2817e-01,  1.1564e-01, -1.6254e-01,  3.0761e-01,
-5.8190e-02,  5.3095e-03, -6.5358e-02, -3.3411e-01, -3.1276e-01,
-1.6786e-01, -1.7184e-01, -1.6713e-01, -4.5754e-03, -2.5831e-01,
 1.1147e-01,  1.1382e-01, -2.1308e-01, -5.0285e-03, -1.5083e-01],
[ 2.7830e-02, -1.3927e-01,  9.6660e-02, -2.4018e-01, -6.5967e-03,
 2.2364e-01,  1.9571e-01,  7.0169e-02,  1.6712e-01, -4.4969e-02,
```

```
          1.4875e-01,  5.6581e-02, -1.1979e-01, -1.1558e-01, -2.3737e-01,
          1.5870e-01, -1.7426e-01, -1.5491e-01, -9.3881e-02,  1.7702e-02],
        [-1.9915e-01, -4.1511e-02, -1.1765e-01, -2.3301e-03,  8.2863e-03,
         -2.9722e-01,  1.3699e-01,  7.1123e-03,  1.9683e-01,  3.7243e-02,
         -1.8939e-01,  7.7163e-02,  9.9315e-02, -8.2109e-02,  1.6666e-01,
          2.1442e-01, -2.4998e-01, -8.3943e-02, -8.4431e-02, -7.1329e-02]],
       device='cuda:0')
------------------------------------------------------------------------------
transformer.layers.0.linear_net.0.bias
tensor([-1.4919e-01, -2.3122e-02,  1.6099e-01,  2.2183e-01,  2.6729e-01,
         2.2939e-01,  3.5927e-03, -3.4440e-03,  7.2553e-02, -2.6878e-03,
         7.4969e-02, -1.1313e-01, -5.9455e-03,  9.7633e-02, -9.0721e-02,
        -1.3349e-01,  8.5191e-02, -1.2266e-01,  9.7176e-03, -1.7770e-01,
        -1.6516e-01, -1.2051e-02,  1.2764e-01, -9.3489e-03,  1.5679e-01,
        -1.4279e-01,  6.9268e-02,  2.0884e-01, -8.9065e-02, -5.6895e-05,
        -1.5753e-01,  1.7756e-01,  1.6528e-01,  1.1669e-01,  7.2080e-02,
         2.4580e-01, -2.9113e-02,  2.2791e-01,  1.5813e-01, -4.4521e-02],
       device='cuda:0')
------------------------------------------------------------------------------
transformer.layers.0.linear_net.3.weight
tensor([[-6.8778e-02, -2.4224e-01,  3.7150e-02,  1.7920e-01,  1.1436e-02,
         -3.7336e-02, -1.4885e-01, -1.0273e-01,  2.0067e-01,  2.2522e-01,
          3.4906e-02, -2.5532e-03, -1.5273e-02, -1.4894e-01, -1.5412e-01,
          3.4929e-02, -2.0323e-03, -9.4760e-02,  1.1391e-01,  1.1694e-01,
          1.6139e-01, -1.0613e-01, -2.4221e-01,  7.3841e-02, -5.5414e-02,
          7.9774e-02, -5.5107e-02,  1.2622e-01, -5.3862e-02,  2.3234e-01,
         -1.1174e-01, -8.8350e-02,  3.3439e-02, -1.4605e-02,  1.1584e-02,
          1.7270e-01,  2.1835e-01, -1.0518e-01, -3.3343e-02,  1.8138e-01],
        [ 1.9439e-01,  1.1570e-01,  1.3796e-01, -1.3893e-01,  1.7136e-01,
          1.7633e-01, -6.7207e-02, -3.0329e-03, -1.7238e-01, -1.7213e-02,
         -8.0629e-03,  5.8940e-02,  8.7595e-02, -4.9755e-02,  2.5491e-02,
          2.9446e-02,  7.9049e-03,  1.8177e-01,  9.3032e-02,  2.2156e-02,
          6.5404e-02, -1.3821e-01,  2.5935e-02, -8.4953e-02,  7.7696e-02,
          1.2929e-01, -1.5839e-01,  1.2302e-01, -1.0843e-03, -6.6546e-02,
          1.7030e-01,  4.9002e-02, -4.5478e-02, -9.9994e-02,  8.8802e-02,
          1.5403e-01, -2.5867e-02,  3.5986e-02,  1.5281e-01, -3.0088e-02],
        [ 1.2461e-01, -4.9068e-02,  2.6652e-02,  1.9453e-01, -2.7050e-02,
         -2.5814e-01,  1.6734e-02, -1.5421e-01, -1.1546e-01, -1.5628e-01,
         -1.0492e-01, -1.5671e-01, -2.8433e-03, -1.5487e-01, -7.8302e-03,
          1.5577e-02,  1.9085e-02,  9.8153e-03, -6.5140e-04,  1.2220e-01,
         -1.4510e-01,  1.4115e-01, -1.2976e-01, -2.0544e-02, -1.2287e-01,
         -6.1976e-02, -2.1606e-01,  1.5984e-01,  1.1266e-01, -1.8133e-01,
          1.8860e-01, -1.3168e-01,  6.5173e-02,  4.4641e-02, -1.2800e-01,
         -1.2737e-01,  2.6048e-01, -1.0482e-01, -8.6706e-02, -5.7421e-02],
        [ 1.0861e-01,  3.1901e-02,  8.5223e-02,  5.8003e-02,  1.8024e-01,
         -4.0771e-02,  3.5821e-02, -1.1779e-02,  1.8776e-03,  3.7282e-02,
         -1.6059e-02,  1.0938e-01,  7.7235e-02, -6.9413e-02,  3.1404e-01,
         -1.1348e-01,  9.8408e-02,  4.0718e-02, -1.6765e-02,  1.2426e-01,
```

```
 -4.0526e-02,  1.1841e-02,  2.6371e-01,  3.4130e-02, -2.2741e-02,
 -1.9136e-02, -1.2615e-01, -4.2318e-03, -1.1093e-01, -2.0960e-01,
  1.2007e-01, -3.1410e-02, -1.6522e-02, -1.5429e-01,  9.8284e-02,
 -2.3132e-02,  6.4800e-03,  1.5780e-01, -2.8369e-02,  2.9173e-02],
[ 4.1703e-02,  2.8313e-02,  6.6820e-02,  3.0845e-02, -2.3005e-02,
  1.7657e-01, -4.5709e-02, -5.8073e-02, -1.8327e-01,  8.2881e-03,
 -1.0970e-01,  1.9527e-01, -1.7280e-01,  5.5413e-02,  5.7735e-02,
  4.8857e-02,  3.6203e-02,  2.9109e-02,  6.1837e-02,  1.1228e-01,
  5.5360e-02, -7.6003e-02,  1.1533e-01,  6.1571e-02,  4.6454e-02,
  3.2075e-02, -8.8575e-02, -6.1707e-02, -1.1745e-03,  1.7276e-01,
  7.6090e-02,  2.9334e-02,  4.6204e-02,  9.2407e-02, -8.6192e-02,
 -3.3578e-02, -1.7094e-01, -4.0046e-02, -2.0491e-01, -1.4010e-01],
[ 1.2066e-01, -4.9175e-02,  1.7092e-01,  1.2244e-01, -1.3144e-05,
  1.8650e-02,  7.1936e-03,  4.4731e-02,  4.8377e-02,  5.4150e-02,
  1.4923e-01, -4.9132e-02, -3.3438e-02, -2.3106e-02,  6.7222e-03,
 -1.7316e-01, -3.3179e-02, -8.6482e-03,  3.2394e-02,  1.9910e-02,
 -3.3746e-02,  3.9182e-02, -1.1631e-01,  4.1492e-02, -1.4931e-01,
 -1.9033e-01, -4.2677e-02, -1.9522e-01,  1.3497e-01, -9.9627e-02,
  4.8909e-02, -2.1210e-01,  8.7635e-02,  3.7552e-02, -9.5787e-02,
 -1.8830e-01,  1.3768e-01, -1.2369e-01, -8.6243e-02,  1.2659e-01],
[-8.7563e-02,  6.2068e-02,  1.0967e-01, -7.3439e-02, -1.5752e-01,
  1.0690e-01, -6.9682e-03,  1.4965e-01,  4.4899e-02,  2.0676e-01,
  2.4843e-02,  1.7340e-02, -1.2870e-01, -1.8834e-01,  8.7644e-02,
  5.7013e-02, -7.2977e-02, -1.1117e-01,  1.2971e-01,  1.4354e-01,
  1.1212e-01, -3.7568e-02, -1.3412e-01, -4.0940e-02,  8.7670e-02,
  1.1678e-01,  1.1919e-01, -1.5328e-02, -1.1324e-01,  4.9621e-03,
 -3.4356e-02,  6.5754e-02,  1.1972e-01, -4.7868e-02,  1.6713e-01,
  7.9939e-02,  1.9990e-01,  5.9917e-02, -1.1743e-01, -7.7393e-02],
[-7.5014e-02, -1.1071e-03,  5.0932e-02, -1.8156e-02,  1.3332e-02,
 -1.2687e-01,  1.0387e-01, -3.9722e-02, -6.3690e-02,  1.7813e-01,
  1.0275e-01,  2.7192e-01, -1.3343e-01,  1.6003e-01, -7.4924e-02,
  1.3909e-01,  2.4546e-02, -1.4190e-01,  1.1960e-01,  1.4524e-01,
  1.8539e-02,  6.7205e-03,  9.7442e-02, -1.0928e-01, -1.4364e-01,
  3.5387e-02, -2.0379e-02,  2.5115e-02,  8.0494e-02,  3.0586e-02,
  2.0625e-01, -1.5262e-01,  7.1132e-02, -1.3741e-01,  4.6208e-02,
 -5.0473e-02,  6.2857e-02, -1.1858e-01,  1.5437e-01,  1.3890e-01],
[-1.9716e-02, -9.2092e-02,  2.5634e-02,  8.9206e-02, -1.1616e-01,
 -1.6873e-01,  1.2904e-01, -1.1102e-02,  2.0180e-01,  9.9228e-02,
 -1.6382e-01, -1.2663e-01,  1.9480e-02,  2.3430e-02,  1.4524e-02,
 -3.4995e-03, -8.3883e-02, -8.3677e-02,  1.5626e-01, -2.0654e-01,
 -3.8002e-02,  8.8679e-02, -8.6282e-02,  1.0716e-01,  7.6333e-02,
 -3.8332e-02,  4.8839e-02, -3.4252e-02,  2.6726e-02,  4.4314e-02,
 -8.4602e-02,  6.8604e-03,  1.4479e-01,  8.7197e-02,  1.2385e-01,
 -1.0809e-01,  1.3881e-01, -2.2268e-01, -1.2708e-01,  1.2384e-01],
[-1.4242e-02, -5.9174e-02, -1.1372e-01, -1.7015e-01,  5.4024e-02,
  1.5848e-01,  8.5317e-02, -4.6928e-02,  9.0350e-02,  1.2259e-01,
 -9.6091e-02, -8.0415e-02,  6.0364e-02,  2.6501e-02, -6.0042e-02,
 -3.4573e-02,  1.6365e-01,  1.1572e-01,  1.2628e-01,  1.1899e-01,
```

```
 -1.2266e-01, -1.3182e-01,  1.1846e-01, -3.8362e-02,  1.6981e-01,
 -4.6808e-02, -1.1595e-01, -3.5294e-02,  5.2302e-02,  3.2769e-02,
  7.3656e-02,  9.0019e-02,  9.0210e-02,  3.9017e-02, -1.4441e-01,
 -1.3337e-01, -1.4087e-01, -4.8085e-02, -8.4509e-02,  7.3603e-02],
[-1.6647e-01,  8.1996e-02,  3.5727e-02,  7.4618e-02,  1.1972e-01,
  2.3961e-01, -1.0421e-01,  2.7473e-03, -1.0076e-01,  2.9853e-02,
  6.2417e-03,  8.6415e-02, -1.5022e-01, -4.0441e-02,  6.5686e-02,
  1.0179e-02, -1.7725e-01,  2.3182e-01, -1.0265e-01,  8.0733e-02,
  1.2839e-01, -2.0426e-01, -2.7475e-01, -1.0304e-01,  1.3011e-01,
 -1.3403e-01,  4.4009e-02,  1.0807e-01, -1.1401e-01,  3.9375e-02,
  5.9146e-02,  3.0880e-03,  1.5813e-01,  1.1565e-01, -5.3711e-02,
 -6.9929e-02,  7.9840e-02, -4.0838e-02,  6.4912e-02,  2.6760e-02],
[ 6.1091e-02,  1.8272e-01, -1.3010e-02, -1.0631e-01,  1.1944e-03,
 -9.9279e-02, -1.4249e-01,  4.8493e-02, -7.4755e-02,  1.6636e-03,
  1.7054e-02,  2.3446e-01, -1.3789e-01,  1.6181e-01, -1.7212e-01,
 -1.1882e-01, -5.1411e-02, -6.3691e-02,  1.3937e-01,  4.5943e-02,
  4.2337e-02, -7.7941e-03,  1.5751e-01, -7.0641e-02, -1.6680e-01,
 -6.9319e-02,  1.6082e-01,  8.7551e-02,  4.3883e-02, -1.3138e-01,
 -1.8072e-01,  4.9180e-02,  8.9904e-02, -4.0644e-02, -1.2237e-01,
  3.7911e-02, -8.5999e-02, -8.2959e-03,  1.4685e-01, -2.8804e-02],
[ 1.7290e-01, -1.5932e-02,  1.9729e-01,  2.6907e-02, -1.7155e-01,
 -7.9671e-02,  1.2444e-01, -6.3479e-02,  7.2451e-02, -7.2894e-02,
  2.9523e-02, -2.2004e-01,  7.6191e-02, -1.8696e-01,  4.8506e-02,
 -1.5097e-01,  1.1234e-01, -2.7872e-03, -6.3315e-02, -7.6310e-02,
  9.7828e-02,  2.8104e-01, -1.4725e-01,  2.0559e-02,  1.1304e-01,
 -1.1089e-01, -1.0163e-01, -1.8537e-01,  1.6125e-01, -5.0447e-02,
 -6.9126e-02,  1.5505e-01,  6.9649e-02, -1.5775e-01,  1.2381e-01,
 -6.7074e-02,  1.6763e-01,  6.4847e-02, -8.8588e-02,  1.3107e-01],
[-1.7926e-01, -7.7552e-02, -1.4982e-01,  5.4436e-03, -2.0323e-01,
 -7.0058e-02,  7.8780e-02,  5.6134e-02,  5.2069e-02,  6.1658e-02,
 -2.2920e-02,  8.6238e-02, -1.0108e-01, -8.9570e-02, -1.4025e-01,
 -5.2642e-02,  1.9674e-02, -5.0850e-02,  8.9190e-02, -4.4458e-03,
 -1.9965e-02,  1.0840e-01, -1.0819e-01,  3.1139e-02,  7.1908e-04,
 -4.4630e-02,  1.8052e-01, -9.1933e-02, -5.9917e-02, -3.1367e-02,
  1.5254e-01, -8.1504e-02,  1.1959e-01,  3.9702e-02,  2.0205e-02,
 -1.1109e-01,  5.9644e-02, -4.8406e-02, -5.0498e-02,  1.8663e-02],
[-1.0417e-03,  3.3601e-02,  1.6945e-02, -5.2869e-02,  5.9415e-02,
  1.7153e-01,  1.9189e-01,  9.2039e-02, -9.3014e-02,  1.9352e-01,
  7.8141e-03, -1.2731e-01, -1.8807e-01,  2.0654e-01,  2.3994e-01,
 -7.4253e-02, -4.1076e-02,  7.4962e-02, -7.7309e-02,  4.5976e-02,
  1.5530e-01,  3.0588e-02,  6.4661e-02,  2.3394e-02,  1.4644e-02,
  1.9254e-02,  1.1210e-01,  2.3306e-01, -1.1304e-01,  1.3587e-01,
 -9.3487e-03, -8.3060e-03,  1.2664e-01,  1.1882e-01,  1.2752e-01,
  9.3114e-02, -1.5820e-01,  4.8531e-02,  1.4287e-01, -4.2457e-02],
[ 2.2584e-02, -6.9310e-02,  3.6619e-03, -1.0067e-01,  4.1722e-03,
 -8.5549e-03, -6.2607e-02, -1.8692e-01, -1.5201e-01, -1.7814e-01,
  8.5204e-02,  1.3946e-01, -2.3507e-01,  4.9203e-02,  3.3754e-02,
  5.2072e-02, -2.8205e-01, -9.6501e-02, -7.1959e-02, -1.6205e-01,
```

```
         -3.5745e-02, -5.0543e-02,  2.2487e-01,  6.5885e-02, -1.2880e-01,
         -1.4834e-02,  5.0688e-03, -9.2862e-02,  8.0278e-02, -1.0247e-01,
          1.0090e-01,  8.1931e-02, -1.0937e-01, -1.4742e-02, -1.9022e-01,
          3.0202e-02, -1.1591e-03,  2.2476e-01, -9.6183e-02,  4.5498e-02],
        [-1.8595e-01,  5.5127e-02,  7.2209e-02,  9.9530e-02, -2.3847e-02,
         -1.3450e-01,  1.0739e-01, -2.1951e-01, -2.6363e-03,  1.1449e-01,
          1.0073e-01, -1.7119e-02, -4.0431e-02, -1.4083e-01, -1.3090e-01,
          2.1945e-02, -1.2374e-01,  1.1058e-01, -1.4417e-01, -1.3120e-01,
         -7.6251e-02,  2.9471e-01, -3.5609e-02, -1.7332e-03, -3.5017e-02,
          1.1529e-01, -8.2926e-02, -8.2471e-02, -1.9134e-01,  1.5420e-01,
          1.3460e-02, -6.9491e-02,  5.0334e-02,  4.8036e-02, -1.4986e-03,
         -5.9858e-02,  1.8359e-01, -5.1070e-02, -1.4628e-01,  3.7749e-02],
        [ 2.9465e-02,  1.4150e-01, -1.1462e-02, -1.2005e-01,  3.6600e-02,
         -7.9199e-02,  8.5128e-02,  7.5485e-02,  6.9996e-02, -2.3606e-01,
          2.5255e-02,  1.2482e-01,  9.7664e-02,  2.4046e-02,  1.0660e-01,
          5.8375e-02,  1.1762e-01,  4.4627e-03,  1.9603e-02, -1.7347e-01,
         -6.7576e-02, -5.6775e-02,  1.2889e-01, -1.7412e-01,  3.9395e-02,
          1.2325e-01, -1.9496e-02, -7.3096e-02, -5.3076e-02, -2.2648e-01,
          4.7900e-02, -8.4568e-02,  2.9301e-02,  3.0221e-03,  1.8876e-02,
         -9.3764e-02,  1.0065e-02, -8.5414e-02, -1.3562e-01,  1.0479e-01],
        [ 6.5128e-02,  2.0297e-01,  5.3280e-02,  5.3287e-02,  3.2529e-03,
          3.6168e-02, -3.0155e-02, -1.3897e-01,  4.1859e-02, -1.5333e-02,
         -5.3221e-02, -4.8780e-02,  8.8297e-02,  9.4832e-02,  5.4373e-02,
          1.3244e-01,  1.1381e-01,  1.4826e-01, -4.5154e-02, -4.1155e-02,
          1.9486e-01, -4.0789e-02, -9.2662e-02,  9.8075e-02,  5.8054e-02,
         -1.3057e-01,  1.5097e-01, -1.1465e-01,  1.1342e-01,  1.1894e-01,
         -1.8292e-01,  5.9793e-02, -5.9557e-02,  2.2603e-02,  1.1528e-01,
         -7.2508e-02, -2.0244e-02, -6.0501e-02, -1.2135e-01, -4.1844e-02],
        [ 6.6775e-02,  1.0714e-02, -4.6353e-02,  4.6640e-02, -1.0653e-01,
         -9.8623e-02,  2.7703e-02, -3.8989e-02,  1.0193e-01, -1.6700e-01,
         -3.6168e-02, -1.2550e-01,  8.6783e-02,  9.0377e-02, -7.6172e-02,
         -2.5866e-02,  1.1293e-01,  1.3784e-01,  1.4807e-01,  1.0437e-01,
         -7.9525e-02, -2.8923e-02, -6.9691e-02,  1.1017e-01,  2.6935e-02,
         -3.9030e-02, -3.9312e-02,  9.0472e-02, -4.3997e-02, -1.6000e-01,
         -1.1767e-01,  2.7378e-02, -3.1754e-02,  3.4474e-02, -1.4763e-01,
         -3.9290e-02, -1.6423e-01,  2.1083e-01, -9.4170e-02,  1.2826e-02]],
       device='cuda:0')
-------------------------------------------------------------------------------
transformer.layers.0.linear_net.3.bias
tensor([ 0.1772,  0.1042, -0.0070, -0.0606, -0.1253,  0.0093,  0.1087, -0.1118,
         0.0570, -0.0111,  0.1154, -0.0812, -0.1314, -0.0398, -0.0337,  0.1071,
         0.0260, -0.1680, -0.1201,  0.1325], device='cuda:0')
-------------------------------------------------------------------------------
transformer.layers.0.norm1.weight
tensor([0.8378, 0.9346, 0.9655, 0.9374, 0.9938, 0.9402, 1.1050, 1.0993, 1.1619,
        1.0821, 1.0516, 0.9468, 0.9618, 1.0136, 1.0113, 1.0311, 0.9660, 1.0284,
        1.0776, 1.0489], device='cuda:0')
-------------------------------------------------------------------------------
```

```
transformer.layers.0.norm1.bias
tensor([ 0.0292,  0.0089,  0.0222,  0.0080, -0.0016,  0.0029,  0.0272, -0.0164,
         0.0020,  0.0066, -0.0232, -0.0488,  0.0105, -0.0039, -0.0296, -0.0061,
        -0.0089, -0.0180,  0.0039,  0.0197], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm2.weight
tensor([0.8659, 0.9843, 0.9372, 0.8484, 1.0230, 0.9267, 1.0689, 1.0865, 1.1448,
        1.0711, 1.0655, 0.8714, 1.0384, 0.9916, 1.0004, 1.1651, 0.9808, 1.0937,
        1.0077, 1.0053], device='cuda:0')
--------------------------------------------------------------------------------
transformer.layers.0.norm2.bias
tensor([ 0.0124,  0.0734, -0.0151,  0.0167, -0.0241, -0.0209,  0.0060,  0.0014,
        -0.0301, -0.0304, -0.0007, -0.0601,  0.0095, -0.0379,  0.0556, -0.0056,
        -0.0231, -0.0298,  0.0462,  0.0068], device='cuda:0')
--------------------------------------------------------------------------------
output_net.0.weight
tensor([[-9.5997e-02, -5.7025e-02, -1.9854e-01,  2.5313e-01,  1.0048e-01,
          6.4003e-02, -5.0339e-03, -5.6503e-02,  5.2796e-02, -2.3143e-01,
         -2.3525e-01, -1.8338e-01,  2.3969e-01, -2.3821e-01,  1.0900e-01,
          2.3849e-02,  4.4907e-02,  1.8865e-01,  8.0013e-02,  1.1968e-01],
        [-1.0732e-01,  1.4386e-02, -6.5414e-02,  1.8589e-02, -7.4050e-02,
         -1.7788e-02, -6.9832e-02,  1.1112e-01, -7.1721e-02, -1.4257e-01,
         -1.1865e-01,  2.9975e-02, -1.0295e-01, -1.4501e-01,  4.3870e-02,
          5.9104e-02, -1.2452e-01,  8.7396e-03, -9.1330e-02, -2.2758e-03],
        [ 2.7073e-02,  1.6152e-01, -7.3733e-02, -2.9963e-02,  6.3972e-02,
         -1.6995e-01, -5.2906e-03,  1.4805e-01, -6.3772e-02,  1.6932e-01,
          5.8848e-02,  1.6335e-01, -2.2808e-01, -2.9985e-01,  2.1954e-01,
          6.8191e-02, -2.0999e-01, -9.4573e-02,  1.1205e-01, -1.0809e-01],
        [ 4.0823e-02, -2.5559e-01, -1.8345e-01, -1.1681e-01, -6.4184e-02,
          8.5115e-02,  2.5085e-02,  2.2204e-02,  3.0492e-02,  1.1962e-01,
         -7.8397e-02,  1.6015e-01, -5.9769e-02, -1.3196e-01, -1.2521e-01,
          1.8879e-02, -5.6805e-02,  3.6420e-01, -5.1108e-02, -2.5591e-01],
        [-9.5082e-02, -6.3274e-02,  1.2639e-02, -4.0070e-02, -3.0388e-02,
          1.9594e-02, -1.6729e-02,  6.5466e-02,  2.8518e-02,  7.2981e-02,
          1.2602e-02,  1.1953e-01,  8.8793e-02,  4.5733e-02, -5.5258e-02,
         -6.4205e-03,  6.2735e-02, -7.5819e-02, -6.8473e-02,  1.4555e-02],
        [ 3.9395e-02, -2.1319e-01,  1.3814e-01,  8.0656e-02,  1.3968e-01,
         -8.6098e-02,  2.1144e-01,  1.3875e-01, -2.1808e-01, -3.5338e-02,
          2.4692e-01,  7.8570e-02, -6.4530e-02,  2.1323e-01,  3.2467e-02,
         -9.2024e-02,  1.3987e-01,  8.5582e-03, -1.8914e-01,  1.7359e-01],
        [-8.1455e-02,  2.7668e-02, -1.1097e-01, -1.1586e-01,  1.0295e-01,
          4.2693e-02, -1.3927e-01, -1.4637e-01, -2.0622e-01,  1.5423e-01,
         -1.0098e-01, -9.5583e-02, -1.8174e-01,  1.0827e-01, -6.2479e-02,
          9.8393e-02, -1.1647e-01,  8.7672e-02,  1.2697e-01,  1.2298e-01],
        [-1.0497e-01,  4.7074e-02, -2.0241e-01,  9.1712e-02,  2.5805e-01,
         -1.1442e-01, -5.5853e-02,  9.6151e-02, -3.6905e-01,  1.7567e-01,
         -1.5954e-01, -2.9755e-02, -2.0960e-01, -5.3763e-02,  3.3597e-02,
          2.2079e-01, -9.0423e-02,  4.1643e-02,  2.4581e-02,  7.9824e-02],
```

```
[ 1.0808e-01, -1.8344e-01,  1.8390e-02, -4.1568e-02,  2.3752e-01,
 -5.7366e-02,  1.5434e-01, -1.1576e-01, -1.1726e-01,  1.2429e-01,
  2.8606e-01,  1.0387e-01,  8.5133e-02, -1.1057e-01, -6.5669e-02,
 -1.2799e-01, -1.9450e-02,  2.9938e-02, -7.5321e-02,  5.8137e-02],
[-6.8611e-05, -3.6722e-02,  4.7494e-02,  6.4983e-02, -4.7102e-02,
  2.6134e-02,  2.0400e-01,  2.7572e-01, -6.2347e-02, -6.3158e-02,
 -7.2471e-02,  8.9716e-02, -7.5281e-02, -2.1225e-02,  1.2725e-01,
  5.2697e-02, -8.8133e-02, -3.0500e-02, -1.5849e-01,  4.9497e-02],
[-1.6123e-01, -6.3785e-02, -1.3328e-01, -2.7431e-02,  1.4174e-02,
  4.1414e-02, -2.4063e-01, -1.5302e-01, -6.8388e-03, -3.7482e-02,
 -3.2390e-02, -2.5030e-02, -6.1990e-02,  4.9512e-02,  3.8850e-02,
 -1.5778e-01, -2.6569e-02,  1.0341e-01,  1.1721e-01,  2.2058e-01],
[-4.9913e-02,  1.7030e-01,  4.3114e-02, -1.6089e-01, -1.3245e-01,
 -3.8794e-02, -3.5636e-02, -4.5575e-03,  8.6139e-02,  7.4966e-02,
 -1.2111e-01,  8.1945e-02, -2.9904e-01, -9.6956e-02,  1.0031e-01,
  5.7046e-02, -1.6024e-01, -1.2120e-01,  1.3894e-01, -9.2676e-02],
[ 4.0087e-02, -3.8802e-02,  2.1559e-01,  7.7049e-02, -6.6093e-02,
  1.7690e-01, -5.9036e-02,  2.5403e-01,  1.2516e-01, -2.1652e-01,
 -2.6109e-01, -1.1008e-02, -1.0978e-01,  2.1819e-02, -1.4476e-01,
  1.2672e-01, -5.8543e-02,  9.9572e-02, -1.1121e-01,  5.8271e-02],
[-1.0446e-01, -1.3008e-01, -1.8100e-02,  1.1230e-01,  1.1653e-01,
  1.5263e-01, -4.2646e-02,  1.8719e-01, -3.7128e-02, -6.0692e-02,
 -1.1640e-01,  2.0820e-01,  1.6376e-01,  7.9044e-02, -1.6798e-01,
  7.4341e-02,  4.2461e-02,  1.5423e-01, -2.0468e-01,  7.4873e-02],
[-2.0395e-01, -3.6301e-03, -1.3445e-02,  8.8830e-03, -1.2365e-01,
 -1.5808e-01,  1.5275e-01, -1.4835e-01,  3.4968e-02, -6.8788e-02,
 -1.2877e-01, -1.6020e-01,  4.2304e-02,  6.9693e-02,  7.5086e-02,
  1.4112e-01,  2.1984e-01, -1.9460e-01, -7.1423e-03, -1.2837e-01],
[ 5.2504e-02, -9.8115e-02,  9.5142e-02, -3.6839e-02,  3.2474e-02,
 -2.9643e-02,  2.4170e-01, -4.5584e-03, -6.4392e-03, -2.3060e-02,
 -1.9630e-02, -1.9849e-01,  2.0396e-01, -1.7321e-01, -6.5392e-02,
  8.0423e-03, -3.8927e-02, -2.3250e-01, -9.8896e-02, -3.2178e-02],
[ 3.6584e-02,  1.0730e-02, -1.5717e-01, -4.6487e-02,  6.0222e-02,
  4.7805e-02,  3.1834e-01, -5.9217e-02, -5.3674e-02,  2.5067e-01,
 -4.2829e-02,  8.6681e-03,  1.1290e-01, -9.6690e-03,  1.4055e-01,
  2.3157e-01, -6.3554e-02,  1.1267e-01,  7.0764e-02, -1.5158e-01],
[-4.9567e-02, -2.1942e-01, -8.3864e-02,  4.8465e-03,  1.2043e-01,
  9.2103e-02,  6.2642e-02,  1.2756e-01,  1.8928e-01,  2.9323e-01,
 -2.2641e-01, -1.2594e-01,  1.9429e-01, -1.6132e-02,  3.9813e-02,
 -1.5988e-01, -8.3924e-02, -1.7130e-01,  5.1869e-02,  1.8470e-01],
[ 5.7644e-02, -2.5278e-01,  1.4257e-01, -1.3545e-01, -3.0134e-01,
 -3.9406e-02,  2.8764e-01,  2.6508e-01,  1.8191e-01,  6.6544e-02,
  3.4709e-02,  2.7410e-02, -1.2490e-01,  7.2273e-02,  1.3802e-01,
 -1.5461e-01, -1.0526e-03, -1.5602e-01, -1.5027e-01, -1.1513e-01],
[-6.8062e-02, -1.7707e-01, -9.1406e-02,  2.2238e-02,  9.8201e-02,
 -2.0011e-01, -1.5366e-01, -1.3102e-01,  5.5296e-02,  7.6381e-02,
  2.4564e-01,  1.5326e-01, -1.1832e-01, -2.5278e-02,  9.1294e-02,
 -2.3433e-01,  1.1913e-01,  9.3920e-02,  8.5807e-02,  4.3786e-02]],
```

```
       device='cuda:0')
--------------------------------------------------------------------------------
output_net.0.bias
tensor([ 2.5755e-01,  1.3733e-01,  3.1281e-02,  1.1338e-01,  7.9566e-02,
        -2.2724e-02, -1.3715e-01, -4.1091e-02, -1.4288e-01, -3.9885e-02,
         7.9907e-02, -1.5303e-01, -1.4051e-01,  1.3282e-01,  1.6469e-01,
        -4.2821e-02,  2.4868e-02, -8.7467e-02,  1.0265e-02,  8.6914e-05],
       device='cuda:0')
--------------------------------------------------------------------------------
output_net.1.weight
tensor([1.3395, 1.2216, 1.3000, 1.2510, 0.9916, 1.2529, 1.2541, 1.2904, 1.2786,
        1.2774, 1.2715, 1.2821, 1.2662, 1.2107, 1.2897, 1.3354, 1.2779, 1.3821,
        1.2726, 1.2521], device='cuda:0')
--------------------------------------------------------------------------------
output_net.1.bias
tensor([0.3203, 0.2338, 0.2468, 0.2269, 0.0113, 0.2341, 0.2318, 0.2501, 0.2379,
        0.2432, 0.2446, 0.2684, 0.2427, 0.2111, 0.2420, 0.3019, 0.2496, 0.3022,
        0.2384, 0.2296], device='cuda:0')
--------------------------------------------------------------------------------
output_net.4.weight
tensor([[-0.2682, -0.2782, -0.0398,  …, -0.3410,  0.3174, -0.3753],
        [ 0.4023,  0.2012, -0.3187,  …, -0.4614, -0.4321, -0.1043],
        [ 0.1181, -0.3251, -0.2473,  …, -0.3905, -0.1780,  0.3544],
        …,
        [-0.0820, -0.1033,  0.0224,  …,  0.3858, -0.2941, -0.2559],
        [-0.1543, -0.0640,  0.1649,  …, -0.3901, -0.3766, -0.2520],
        [-0.4253, -0.0671,  0.0097,  …,  0.4040,  0.5006,  0.0212]],
       device='cuda:0')
--------------------------------------------------------------------------------
output_net.4.bias
tensor([-0.0247,  0.1526,  0.0850, -0.1464,  0.1352,  0.0385,  0.0652, -0.1488,
         0.1002,  0.1606, -0.0224,  0.1837, -0.1837, -0.1388, -0.1809,  0.1591,
         0.0405,  0.0793, -0.2122, -0.0332,  0.3101, -0.0276, -0.0770, -0.1425,
         0.1671, -0.0681,  0.0766, -0.1864,  0.1755, -0.1156,  0.1637, -0.1354,
        -0.0619, -0.3599, -0.0642,  0.2843,  0.0645, -0.2369,  0.1238, -0.0381,
         0.0584,  0.2338, -0.0972, -0.1162,  0.0536,  0.0171,  0.2094,  0.0089,
         0.0999,  0.0712, -0.1499, -0.0462, -0.0497,  0.2528, -0.0414, -0.0873,
        -0.0703, -0.0708,  0.1032,  0.0742, -0.1147, -0.0676, -0.2692, -0.0036,
        -0.1610,  0.0314, -0.1036,  0.0444,  0.1313, -0.0321,  0.1774,  0.2564,
        -0.2026, -0.0478,  0.0321, -0.0313, -0.2394,  0.1427, -0.1704, -0.0854,
        -0.0810,  0.0006,  0.0310,  0.2747,  0.0530, -0.2046,  0.1749, -0.1904,
         0.0803, -0.0581, -0.3051, -0.2040,  0.0271,  0.1774,  0.1685, -0.1951,
         0.1281,  0.1445,  0.0982, -0.2242], device='cuda:0')
--------------------------------------------------------------------------------
```

```
[ ]:
```