**Code:**

import torch

import torch.nn as nn

import torch.nn.functional as F

import math

class PositionalEncoding(nn.Module):

    def \_\_init\_\_(self, d\_model, max\_len=5000):

        super(PositionalEncoding, self).\_\_init\_\_()

        self.dropout = nn.Dropout(p=0.1)

        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).transpose(0, 1)

        self.register\_buffer('pe', pe)

    def forward(self, x):

        x = x + self.pe[:x.size(0), :]

        return self.dropout(x)

class Transformer(nn.Module):

    def \_\_init\_\_(self, vocab\_size, d\_model, nhead, num\_encoder\_layers, num\_decoder\_layers, dim\_feedforward, max\_seq\_length):

        super(Transformer, self).\_\_init\_\_()

        self.embedding = nn.Embedding(vocab\_size, d\_model)

        self.pos\_encoder = PositionalEncoding(d\_model, max\_seq\_length)

        self.transformer = nn.Transformer(d\_model=d\_model, nhead=nhead, num\_encoder\_layers=num\_encoder\_layers,

                                           num\_decoder\_layers=num\_decoder\_layers, dim\_feedforward=dim\_feedforward)

        self.fc = nn.Linear(d\_model, vocab\_size)

    def forward(self, src, tgt):

        src = self.embedding(src)

        src = self.pos\_encoder(src)

        tgt = self.embedding(tgt)

        tgt = self.pos\_encoder(tgt)

        output = self.transformer(src, tgt)

        output = self.fc(output)

        return output

# Example usage:

# Define model parameters

vocab\_size = 10000

d\_model = 512

nhead = 8

num\_encoder\_layers = 6

num\_decoder\_layers = 6

dim\_feedforward = 2048

max\_seq\_length = 100

# Create the model

model = Transformer(vocab\_size, d\_model, nhead, num\_encoder\_layers, num\_decoder\_layers, dim\_feedforward, max\_seq\_length)

# Define some input tensors (for example purposes)

src = torch.randint(0, vocab\_size, (10, max\_seq\_length))

tgt = torch.randint(0, vocab\_size, (10, max\_seq\_length))

# Pass inputs through the model

output = model(src, tgt)

print(output.shape)

**Output:**

****