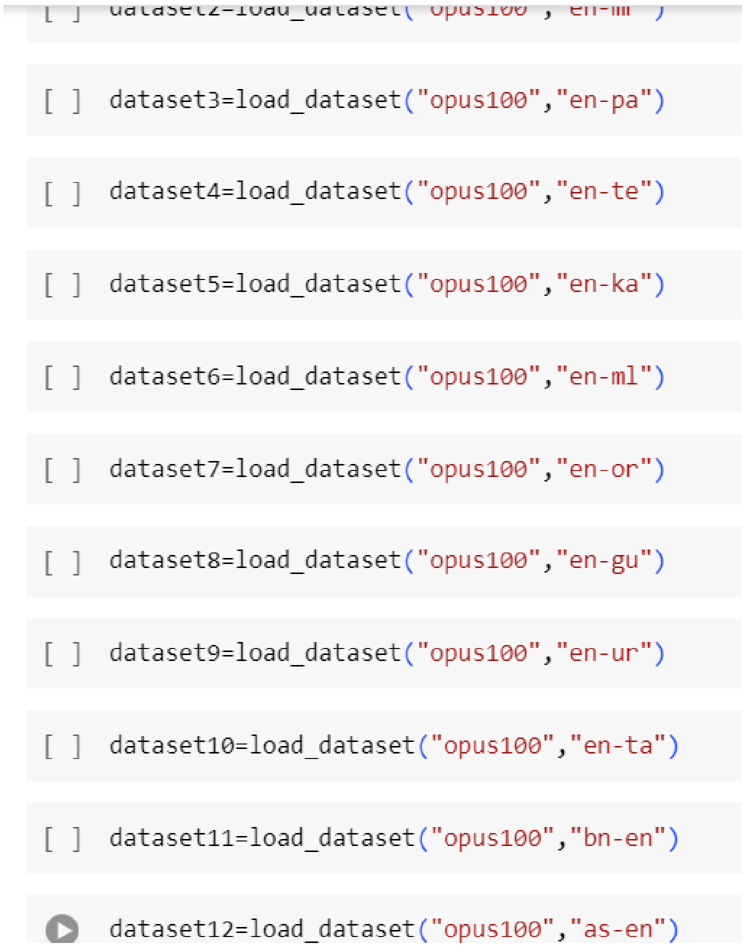
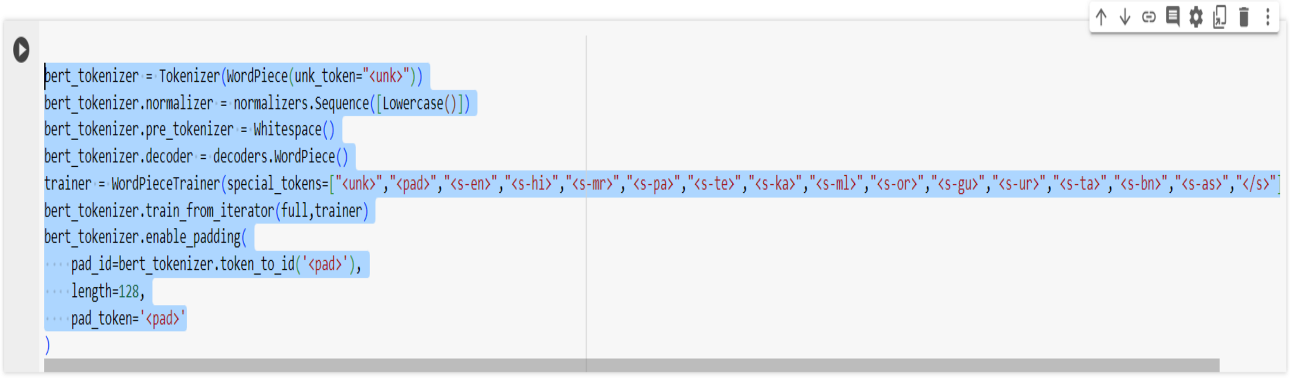
Good afternoon everyone…. Our project topic is multilingual translator using transformer nd my project members are ashi and Aishwarya….The goal of our project is to develop a system that can accurately translate a text between multiple languages.transformer is a deep learning model based on self attention mechanism which is giving high translational accuracy for various language pairs. Here we have tried to create sequence to sequence transformer architecture from scratch.



```
dataset2=load_dataset("opus100","en-hi")
dataset3=load_dataset("opus100","en-pa")
dataset4=load_dataset("opus100","en-te")
dataset5=load_dataset("opus100","en-ka")
dataset6=load_dataset("opus100","en-ml")
dataset7=load_dataset("opus100","en-or")
dataset8=load_dataset("opus100","en-gu")
dataset9=load_dataset("opus100","en-ur")
dataset10=load_dataset("opus100","en-ta")
dataset11=load_dataset("opus100","bn-en")
dataset12=load_dataset("opus100","as-en")
```

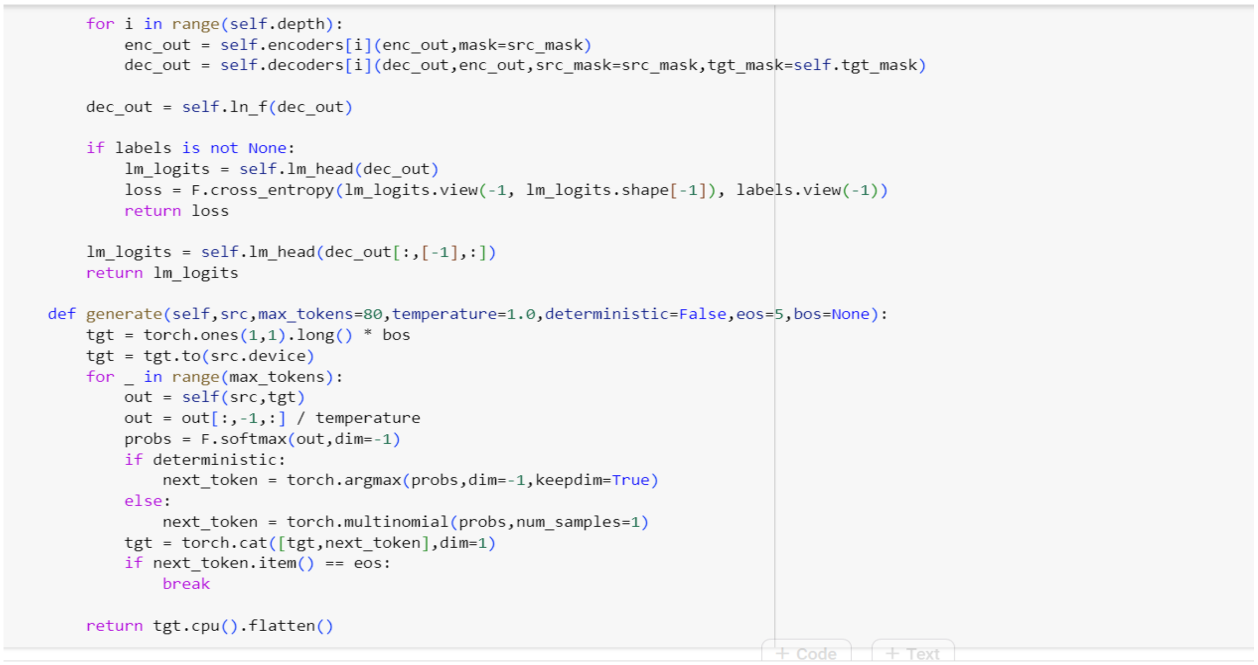
Here we have collected the dataset from opus100 which contains translation pairs .

```
+--------------------------------------------------------------------------------+
```



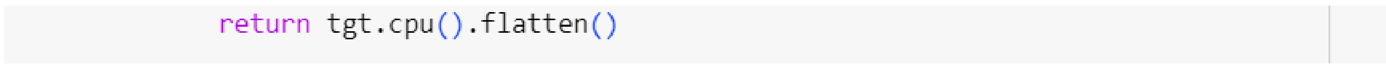
```
bert_tokenizer = Tokenizer(WordPiece(unk_token="<unk>"))
bert_tokenizer.normalizer = normalizers.Sequence([Lowercase()])
bert_tokenizer.pre_tokenizer = Whitespace()
bert_tokenizer.decoder = decoders.WordPiece()
trainer = WordPieceTrainer(special_tokens=["<unk>","<pad>","<s-en>","<s-hi>","<s-mr>","<s-pa>","<s-te>","<s-ka>","<s-ml>","<s-or>","<s-gu>","<s-ur>","<s-ta>","<s-bn>","<s-as>","</s>"
bert_tokenizer.train_from_iterator(full,trainer)
bert_tokenizer.enable_padding(
    pad_id=bert_tokenizer.token_to_id('<pad>'),
    length=128,
    pad_token='<pad>'
)
```

We used models like BERT for text tokenization process .This process splits the words into tokens based on whitespaces. the no of maximum words in a sentence are 128.

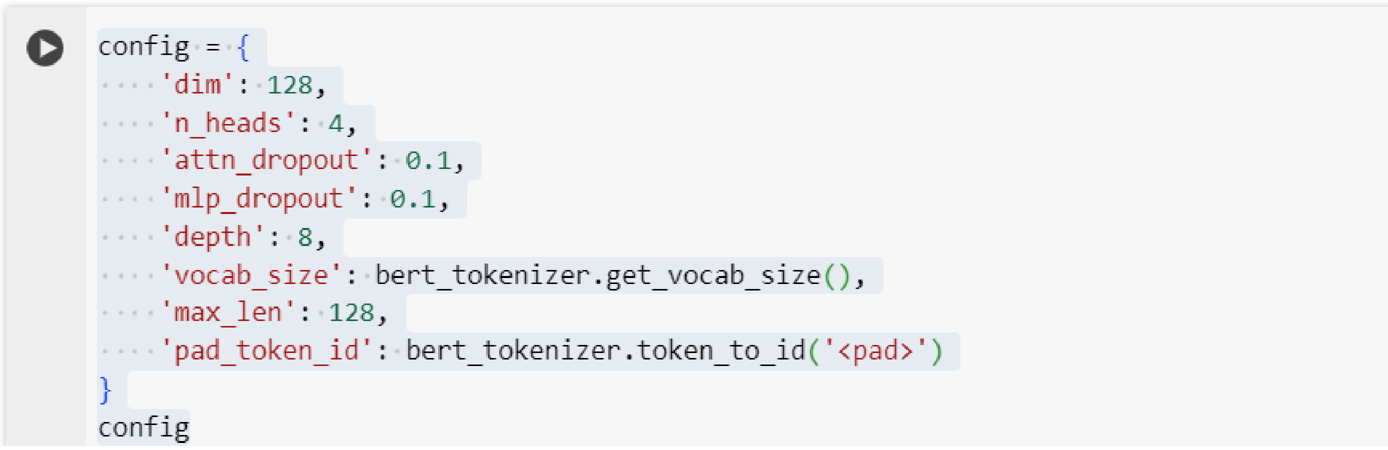


```
        for i in range(self.depth):
            enc_out = self.encoders[i](enc_out,mask=src_mask)
            dec_out = self.decoders[i](dec_out,enc_out,src_mask=src_mask,tgt_mask=self.tgt_mask)

        dec_out = self.ln_f(dec_out)

        if labels is not None:
            lm_logits = self.lm_head(dec_out)
            loss = F.cross_entropy(lm_logits.view(-1, lm_logits.shape[-1]), labels.view(-1))
            return loss

        lm_logits = self.lm_head(dec_out[:,[-1],:])
        return lm_logits

    def generate(self,src,max_tokens=80,temperature=1.0,deterministic=False,eos=5,bos=None):
        tgt = torch.ones(1,1).long() * bos
        tgt = tgt.to(src.device)
        for _ in range(max_tokens):
            out = self(src,tgt)
            out = out[:,-1,:] / temperature
            probs = F.softmax(out,dim=-1)
            if deterministic:
                next_token = torch.argmax(probs,dim=-1,keepdim=True)
            else:
                next_token = torch.multinomial(probs,num_samples=1)
            tgt = torch.cat([tgt,next_token],dim=1)
            if next_token.item() == eos:
                break

        return tgt.cpu().flatten()
```

This is whole transformer architecture .

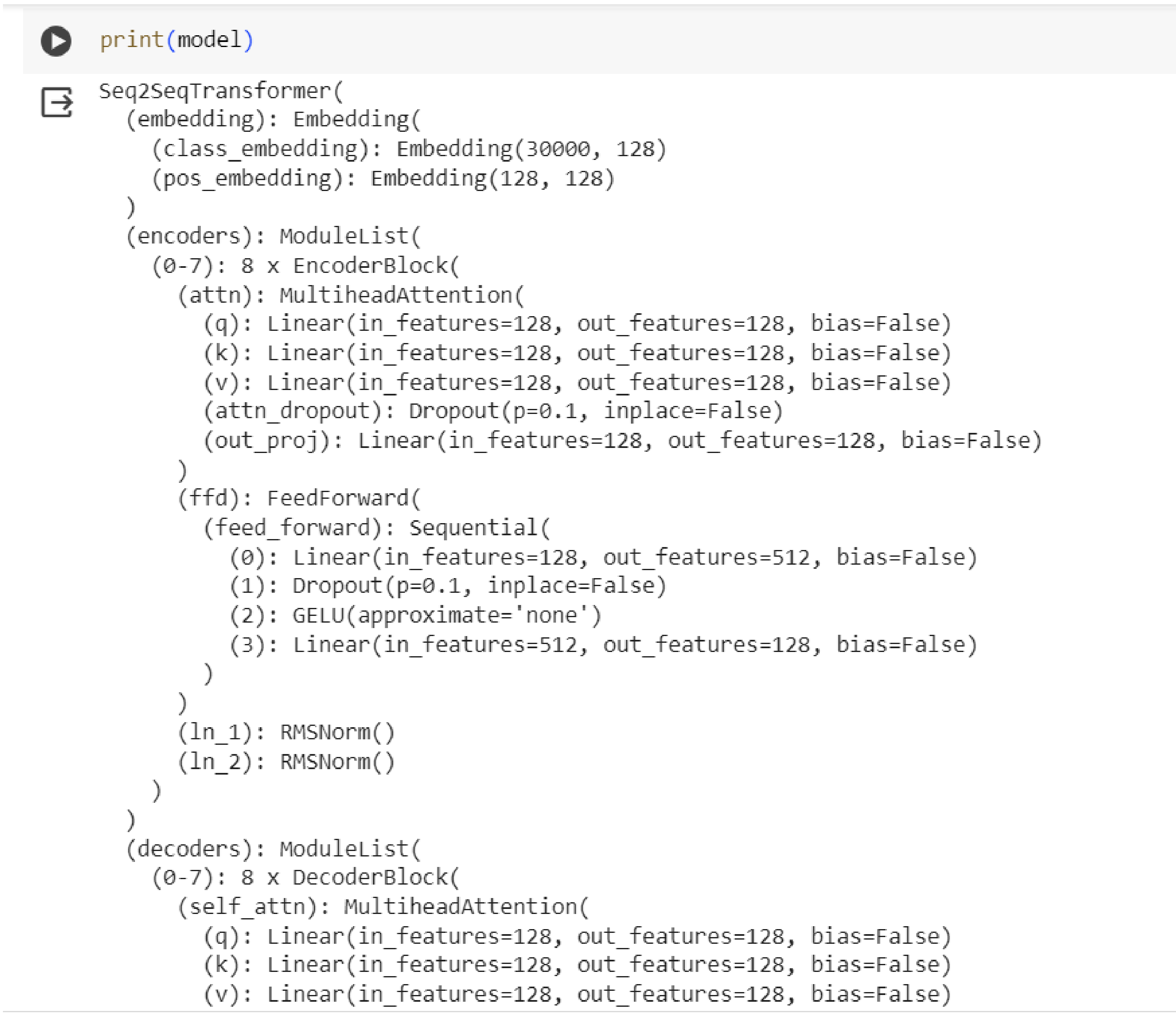


```
            return tgt.cpu().flatten()
```



```
config = {
    'dim': 128,
    'n_heads': 4,
    'attn_dropout': 0.1,
    'mlp_dropout': 0.1,
    'depth': 8,
    'vocab_size': bert_tokenizer.get_vocab_size(),
    'max_len': 128,
    'pad_token_id': bert_tokenizer.token_to_id('<pad>')
}
config
```

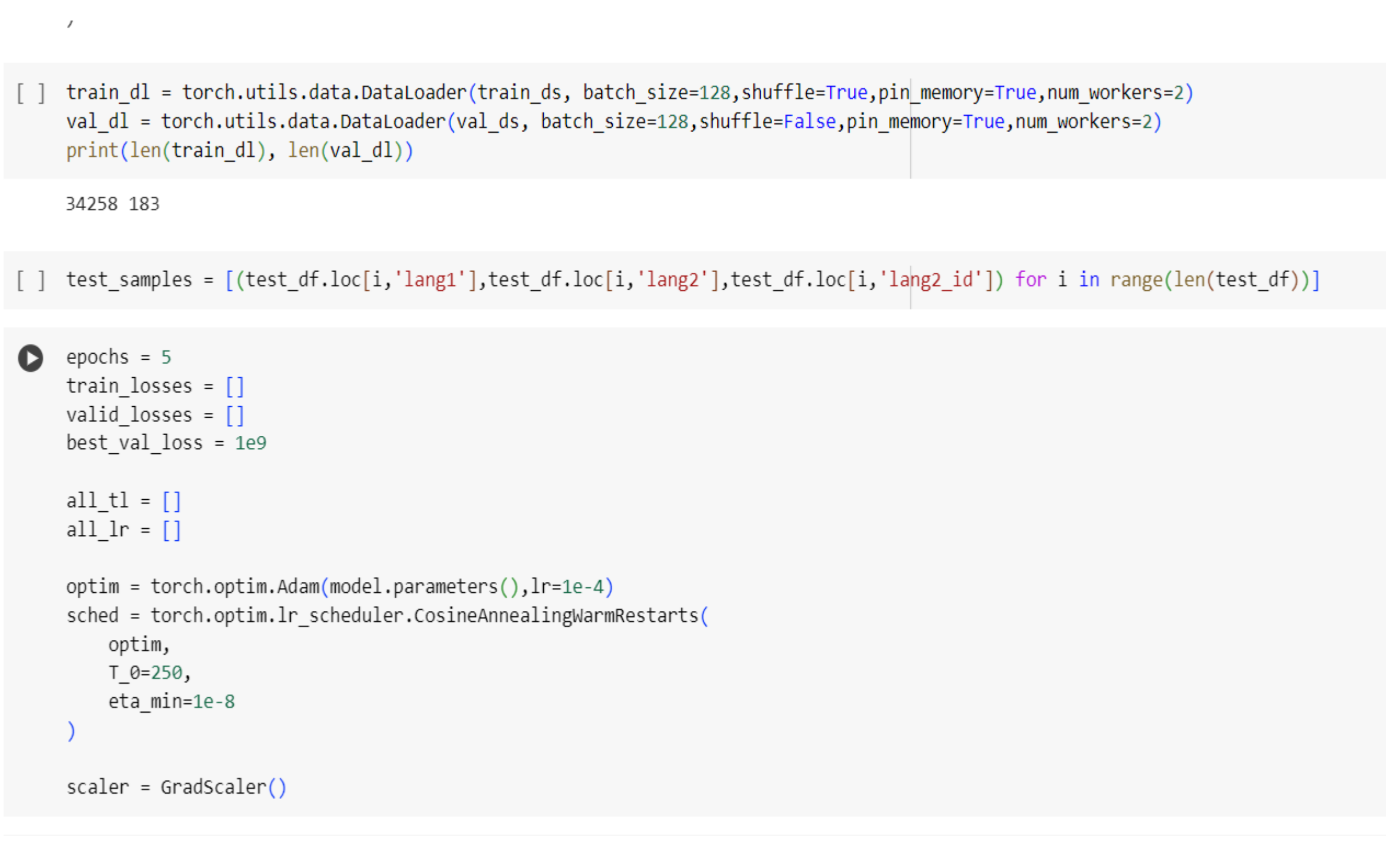
Here , no of maximum words in a sentence are 128 , no of heads are 4 and for tokenization process we have used bert tokenization.



```
print(model)
```

```
Seq2SeqTransformer(
  (embedding): Embedding(
    (class_embedding): Embedding(30000, 128)
    (pos_embedding): Embedding(128, 128)
  )
  (encoders): ModuleList(
    (0-7): 8 x EncoderBlock(
      (attn): MultiheadAttention(
        (q): Linear(in_features=128, out_features=128, bias=False)
        (k): Linear(in_features=128, out_features=128, bias=False)
        (v): Linear(in_features=128, out_features=128, bias=False)
        (attn_dropout): Dropout(p=0.1, inplace=False)
        (out_proj): Linear(in_features=128, out_features=128, bias=False)
      )
      (ffd): FeedForward(
        (feed_forward): Sequential(
          (0): Linear(in_features=128, out_features=512, bias=False)
          (1): Dropout(p=0.1, inplace=False)
          (2): GELU(approximate='none')
          (3): Linear(in_features=512, out_features=128, bias=False)
        )
      )
      (ln_1): RMSNorm()
      (ln_2): RMSNorm()
    )
  )
  (decoders): ModuleList(
    (0-7): 8 x DecoderBlock(
      (self_attn): MultiheadAttention(
        (q): Linear(in_features=128, out_features=128, bias=False)
        (k): Linear(in_features=128, out_features=128, bias=False)
        (v): Linear(in_features=128, out_features=128, bias=False)
```

This is our model summary.



```
,
```

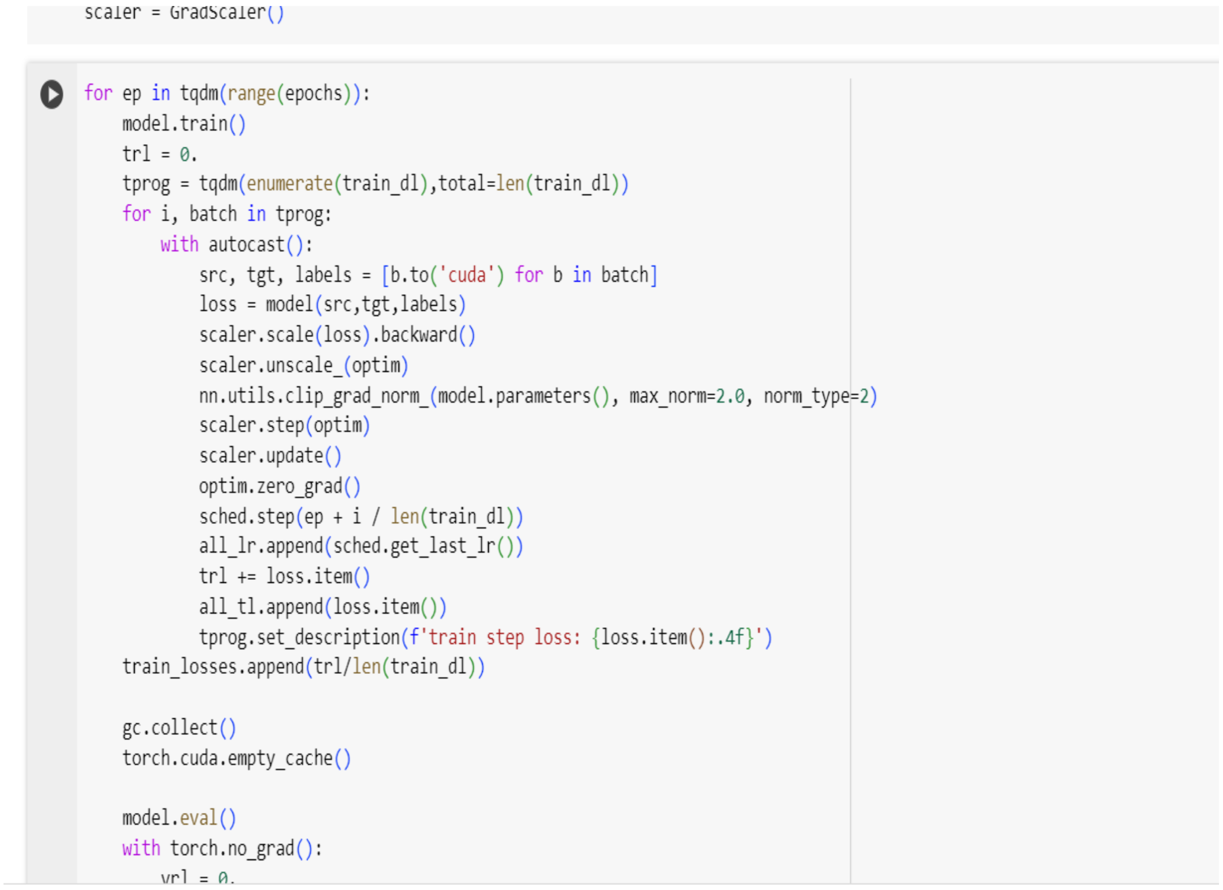
```
[ ] train_dl = torch.utils.data.DataLoader(train_ds, batch_size=128,shuffle=True,pin_memory=True,num_workers=2)
    val_dl = torch.utils.data.DataLoader(val_ds, batch_size=128,shuffle=False,pin_memory=True,num_workers=2)
    print(len(train_dl), len(val_dl))
```

```
34258 183
```

```
[ ] test_samples = [(test_df.loc[i,'lang1'],test_df.loc[i,'lang2'],test_df.loc[i,'lang2_id']) for i in range(len(test_df))]
```

```
epochs = 5
train_losses = []
valid_losses = []
best_val_loss = 1e9

all_tl = []
all_lr = []

optim = torch.optim.Adam(model.parameters(),lr=1e-4)
sched = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(
    optim,
    T_0=250,
    eta_min=1e-8
)

scaler = GradScaler()
```

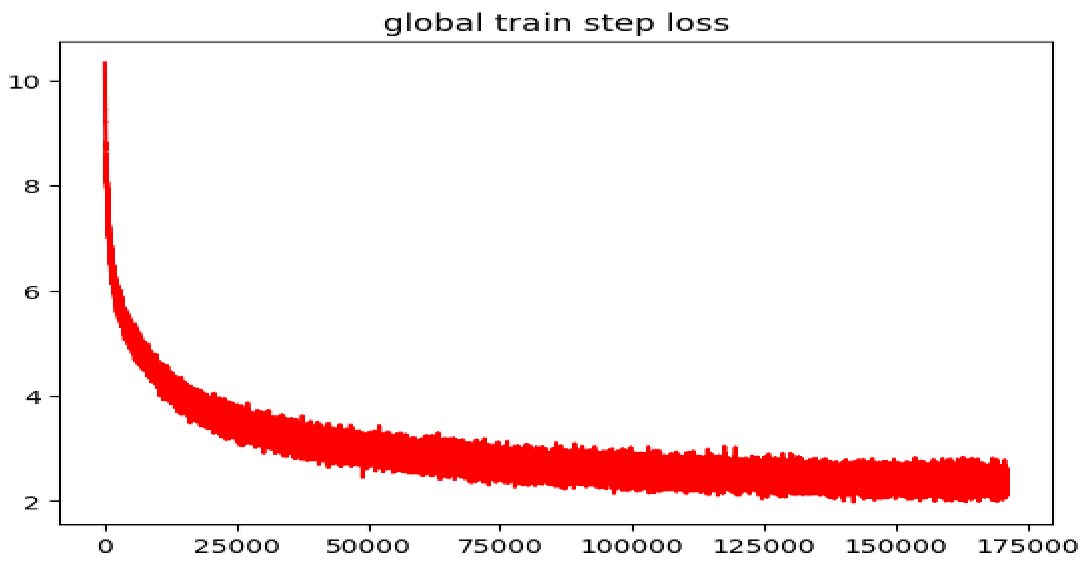
We gave 5 no of epochs. Then find the train and validation loss for each epoch with smallest learning rate . We have used adam optimizer for updation of models weights. Gradscaler for scaling the loss and to increase training speed.It will also reduce memory consumption.



```
scaler = GradScaler()
```

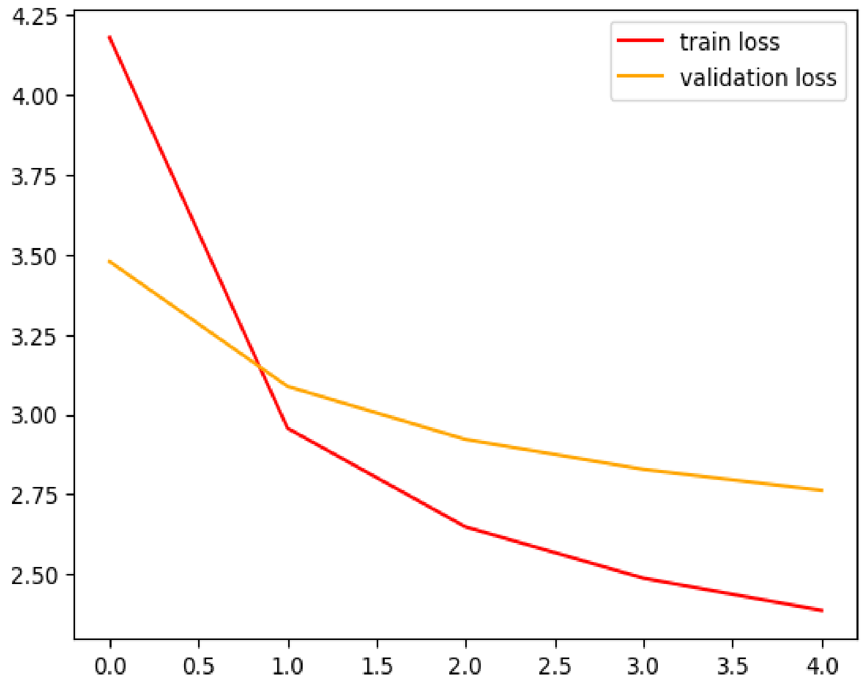
```
for ep in tqdm(range(epochs)):
    model.train()
    trl = 0.
    tprog = tqdm(enumerate(train_dl),total=len(train_dl))
    for i, batch in tprog:
        with autocast():
            src, tgt, labels = [b.to('cuda') for b in batch]
            loss = model(src,tgt,labels)
            scaler.scale(loss).backward()
            scaler.unscale_(optim)
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0, norm_type=2)
            scaler.step(optim)
            scaler.update()
            optim.zero_grad()
            sched.step(ep + i / len(train_dl))
            all_lr.append(sched.get_last_lr())
            trl += loss.item()
            all_tl.append(loss.item())
            tprog.set_description(f'train step loss: {loss.item():.4f}')
    train_losses.append(trl/len(train_dl))

    gc.collect()
    torch.cuda.empty_cache()

    model.eval()
    with torch.no_grad():
        vrl = 0.
```

This is the training loop for translation model using pytorch. If validational loss is less then we have saved the model. We called gc.collect() to free unused memory.



y-axis = loss value

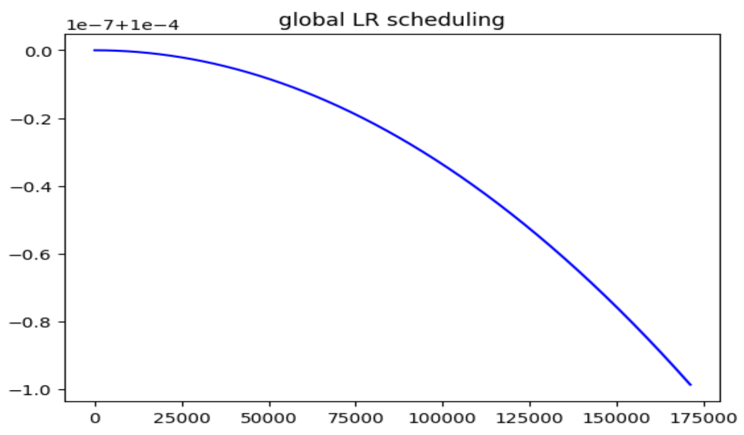
x-axis = no of training iterations



x-axis = no of epochs

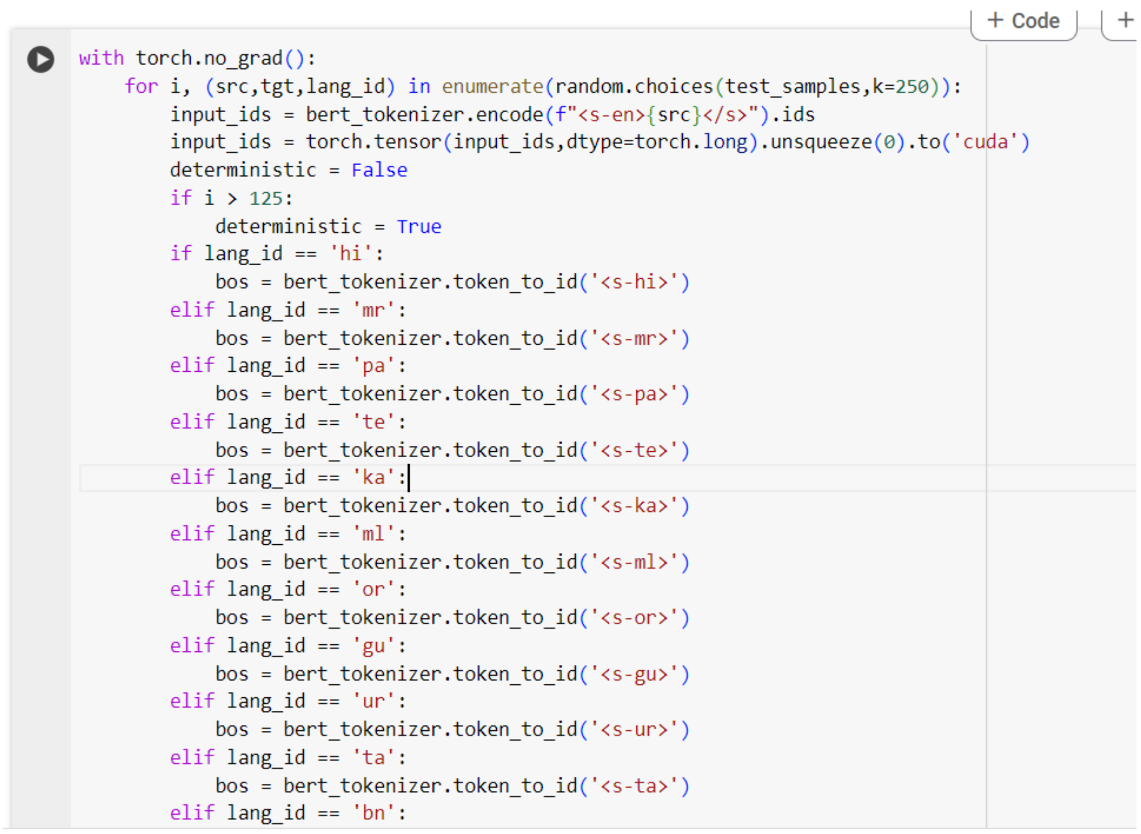
y- axis = loss value

the model seems to perform well up to the fourth epoch, with both training and validation losses decreasing.



y-axis = learning rate values

x- axis = no of training steps



```python
with torch.no_grad():
    for i, (src,tgt,lang_id) in enumerate(random.choices(test_samples,k=250)):
        input_ids = bert_tokenizer.encode(f"<s-en>{src}</s>").ids
        input_ids = torch.tensor(input_ids,dtype=torch.long).unsqueeze(0).to('cuda')
        deterministic = False
        if i > 125:
            deterministic = True
        if lang_id == 'hi':
            bos = bert_tokenizer.token_to_id('<s-hi>')
        elif lang_id == 'mr':
            bos = bert_tokenizer.token_to_id('<s-mr>')
        elif lang_id == 'pa':
            bos = bert_tokenizer.token_to_id('<s-pa>')
        elif lang_id == 'te':
            bos = bert_tokenizer.token_to_id('<s-te>')
        elif lang_id == 'ka':
            bos = bert_tokenizer.token_to_id('<s-ka>')
        elif lang_id == 'ml':
            bos = bert_tokenizer.token_to_id('<s-ml>')
        elif lang_id == 'or':
            bos = bert_tokenizer.token_to_id('<s-or>')
        elif lang_id == 'gu':
            bos = bert_tokenizer.token_to_id('<s-gu>')
        elif lang_id == 'ur':
            bos = bert_tokenizer.token_to_id('<s-ur>')
        elif lang_id == 'ta':
            bos = bert_tokenizer.token_to_id('<s-ta>')
        elif lang_id == 'bn':
```

We have turned off gradient computations for saving memory and giving 250 test examples.