原論文に従ってemb\_dimの4倍をデフォルト値としている

215

dropout: ドロップアウト率

216

"""

217

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

218

# 1つ目のLayer Normalization [2-5-2項]

219

self.ln1 = nn.LayerNorm(emb\_dim)

220

# MHSA [2-4-7項]

221

self.msa = MultiHeadSelfAttention(

222

emb\_dim=emb\_dim, head=head,

223

dropout = dropout,

224

)

225

# 2つ目のLayer Normalization [2-5-2項]

226

self.ln2 = nn.LayerNorm(emb\_dim)

227

# MLP [2-5-3項]

228

self.mlp = nn.Sequential(

229

nn.Linear(emb\_dim, hidden\_dim),

230

nn.GELU(),

231

nn.Dropout(dropout),

232

nn.Linear(hidden\_dim, emb\_dim),

233

nn.Dropout(dropout)

234

)

235

def forward(self, z: torch.Tensor) -> torch.Tensor:

236

"""

237

引数:

238

z: Encoder Blockへの入力。形状は、(B, N, D)

239

B: バッチサイズ、N:トークンの数、D:ベクトルの長さ

240

241

返り値:

242

out: Encoder Blockへの出力。形状は、(B, N, D)。[式(10)]

243

B:バッチサイズ、N:トークンの数、D:埋め込みベクトルの長さ

244

"""

245

# Encoder Blockの前半部分 [式(12)]

246

out = self.msa(self.ln1(z)) + z

247

# Encoder Blockの後半部分 [式(13)]

248

out = self.mlp(self.ln2(out)) + out

249

return out

250

251

vit\_enc = VitEncoderBlock()

252

z\_1 = vit\_enc(z\_0) #z\_0は2-2節のz\_0=input\_layer(x)で、形状は(B, N, D)

253

254

# (2, 5, 384)(=(B, N, D))になっていることを確認

255

print(z\_1.shape)

256

257

258

259

# ----------------------------

260

# 2-6 ViTの実装

261

# ----------------------------

262

print("=======2-6 ViTの実装=======")

263

264

class Vit(nn.Module):

265

def \_\_init\_\_(self, in\_channels:int=3, num\_classes:int=10, emb\_dim:int=384, num\_patch\_row:int=2, image\_size:int=32, num\_blocks:int=7, head:int=8, hidden\_dim:int=384\*4, dropout:float=0.):

266

"""

267

引数:

268

in\_channels: 入力画像のチャンネル数

269

num\_classes: 画像分類のクラス数

270

emb\_dim: 埋め込み後のベクトルの長さ

271

num\_patch\_row: 1辺のパッチの数

272

image\_size: 入力画像の1辺の大きさ。入力画像の高さと幅は同じであると仮定

273

num\_blocks: Encoder Blockの数

274

head: ヘッドの数

275

hidden\_dim: Encoder BlockのMLPにおける中間層のベクトルの長さ

276

dropout: ドロップアウト率

277

"""

278

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

279

# Input Layer [2-3節]

280

self.input\_layer = VitInputLayer(

281

in\_channels,

282

emb\_dim,

283

num\_patch\_row,

284

image\_size)

285

286

# Encoder。Encoder Blockの多段。[2-5節]

287

self.encoder = nn.Sequential(\*[

288

VitEncoderBlock(

289

emb\_dim=emb\_dim,

290

head=head,

291

hidden\_dim=hidden\_dim,

292

dropout = dropout

293

)

294

for \_ in range(num\_blocks)])

295

296

# MLP Head [2-6-1項]

297

self.mlp\_head = nn.Sequential(

298

nn.LayerNorm(emb\_dim),

299

nn.Linear(emb\_dim, num\_classes)

300

)

301

302

def forward(self, x: torch.Tensor) -> torch.Tensor:

303

"""

304

引数:

305

x: ViTへの入力画像。形状は、(B, C, H, W)

306

B: バッチサイズ、C:チャンネル数、H:高さ、W:幅

307

308

返り値:

309

out: ViTの出力。形状は、(B, M)。[式(10)]

310

B:バッチサイズ、M:クラス数

311

"""

312

# Input Layer [式(14)]

313

## (B, C, H, W) -> (B, N, D)

314

## N: トークン数(=パッチの数+1), D: ベクトルの長さ

315

out = self.input\_layer(x)

316

317

# Encoder [式(15)、式(16)]

318

## (B, N, D) -> (B, N, D)

319

out = self.encoder(out)

320

321

# クラストークンのみ抜き出す

322

## (B, N, D) -> (B, D)

323

cls\_token = out[:,0]

324

325

# MLP Head [式(17)]

326

## (B, D) -> (B, M)

327

pred = self.mlp\_head(cls\_token)

328

return pred

329

330

num\_classes = 10

331

batch\_size, channel, height, width= 2, 3, 32, 32

332

x = torch.randn(batch\_size, channel, height, width)

333

vit = Vit(in\_channels=channel, num\_classes=num\_classes)

334

pred = vit(x)

335

336

# (2, 10)(=(B, M))になっていることを確認

337

print(pred.shape)