Attention, Transformer, ViT

Implementation details overview

Andrey Stotskiy



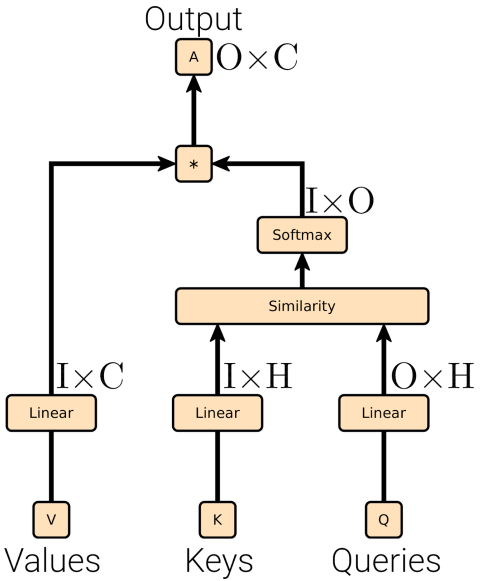






October 2026

Attention



I - input sequence length

O - output sequence length

H - key & query dimensionality

C - value & output dimensionality

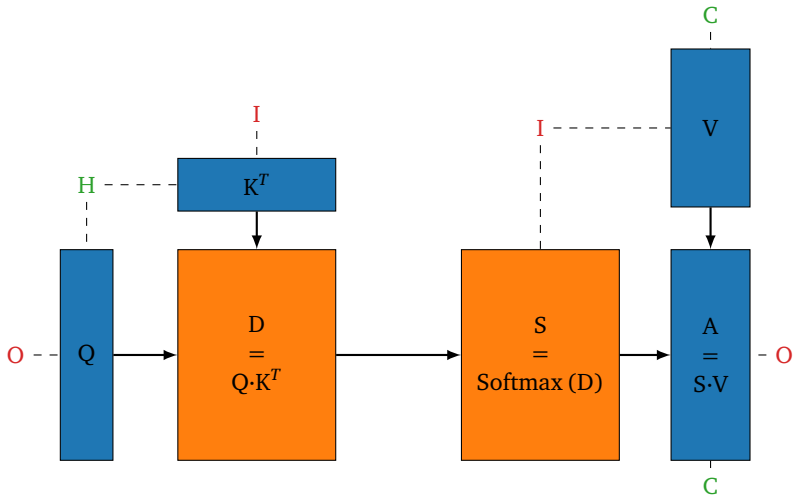
Linear applied over last dimension (H/C)

Similarity is a measure of how close each Q is to each K (usually, dot product is used)

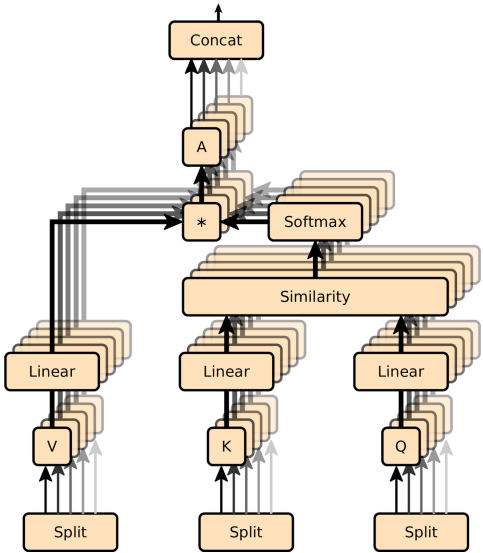
Softmax is applied along the I dimension

$*$ is matrix multiplication

Attention

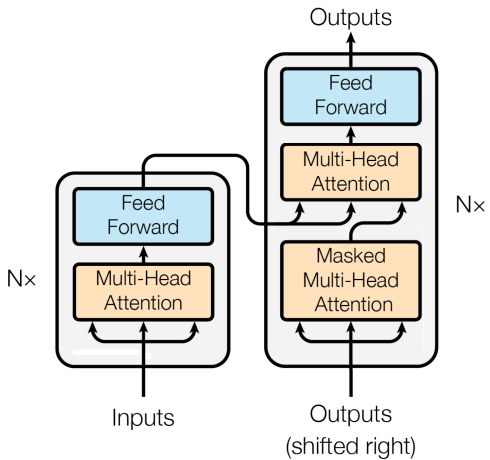


Multi-head Attention



1. Split the inputs into h feature groups (usually, $\mathrm{h} = 8$)
2. Compute attention separately in each group
3. Concatenate the results from all groups

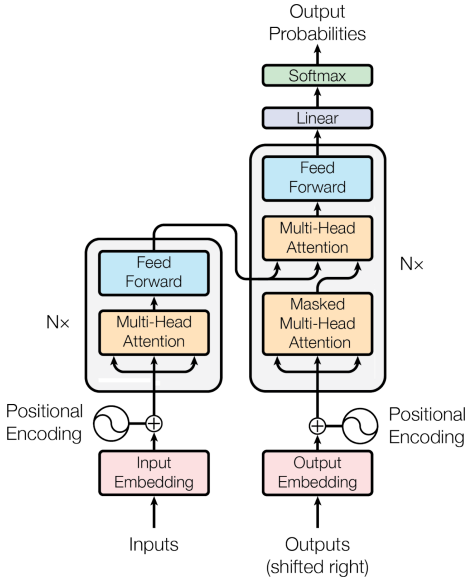
Transformer



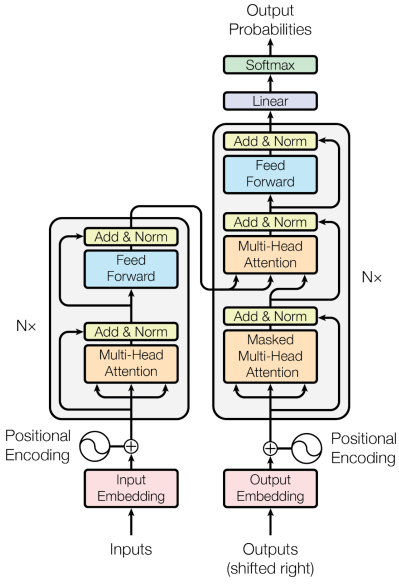
Feed Forward = Linear, ReLU, Linear

Repeat each block N times (N = 6)

Transformer



Transformer



Vision Transformer

```
class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout = 0.):
        super().__init__()
        self.net = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, hidden_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, dim),
            nn.Dropout(dropout)
        )

    def forward(self, x):
        return self.net(x)
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Vision Transformer

```
class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head *  heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.norm = nn.LayerNorm(dim)

        self.attend = nn.Softmax(dim = -1)
        self.dropout = nn.Dropout(dropout)

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()
    ...
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Vision Transformer

```
class Attention(nn.Module):
    ...
    def forward(self, x):
        x = self.norm(x)

        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = self.heads), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)
        attn = self.dropout(attn)

        out = torch.matmul(attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Vision Transformer

```
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.norm = nn.LayerNorm(dim)
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Attention(dim, heads = heads, dim_head = dim_head, dropout = dropout),
                FeedForward(dim, mlp_dim, dropout = dropout)
            ]))

    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x

        return self.norm(x)
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Vision Transformer

```
class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_dim,
                 pool = 'cls', channels = 3, dim_head = 64, dropout = 0., emb_dropout = 0.):
        ...
        self.to_patch_embedding = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = patch_height, p2 = patch_width),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, dim),
            nn.LayerNorm(dim),
        )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool
        self.to_latent = nn.Identity()

        self.mlp_head = nn.Linear(dim, num_classes)
    ...
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Vision Transformer

```
class ViT(nn.Module):
    ...
    def forward(self, img):
        x = self.to_patch_embedding(img)
        b, n, _ = x.shape

        cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b = b)
        x = torch.cat((cls_tokens, x), dim=1)
        x += self.pos_embedding[:, :(n + 1)]
        x = self.dropout(x)

        x = self.transformer(x)

        x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

        x = self.to_latent(x)
        return self.mlp_head(x)
```

github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py