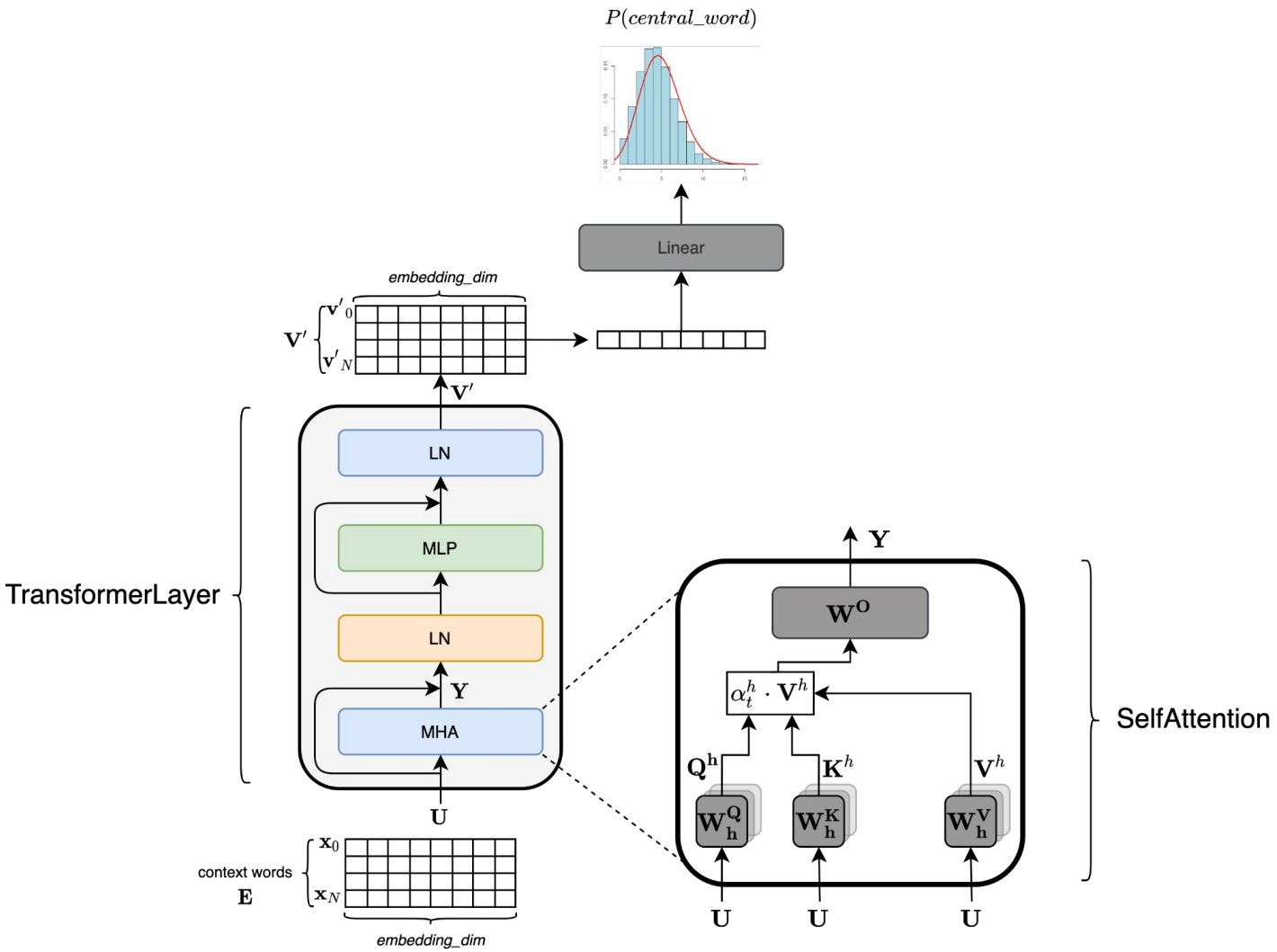
# Assignment 2
# Language Modeling

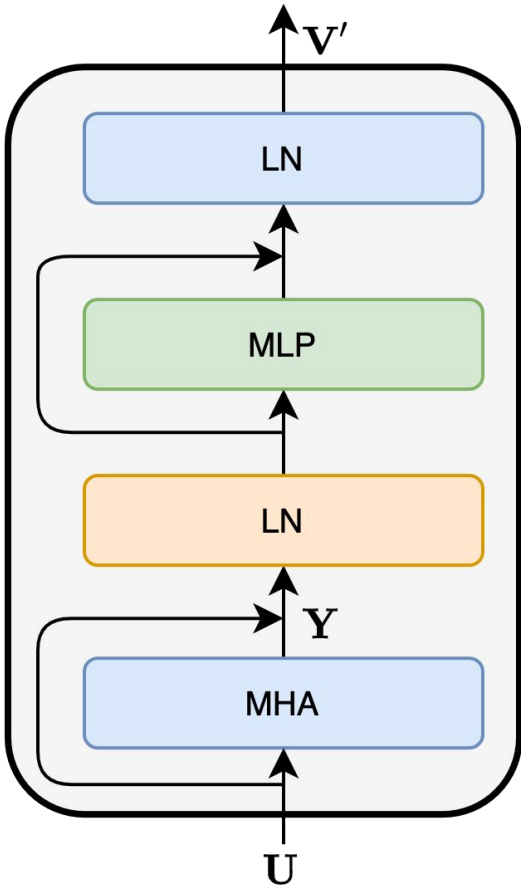
SLPDL 2022

Javier Ferrando Monsonís
javier.ferrando.mosnsonis@upc.edu

# Task: Improve the Transformer Baseline



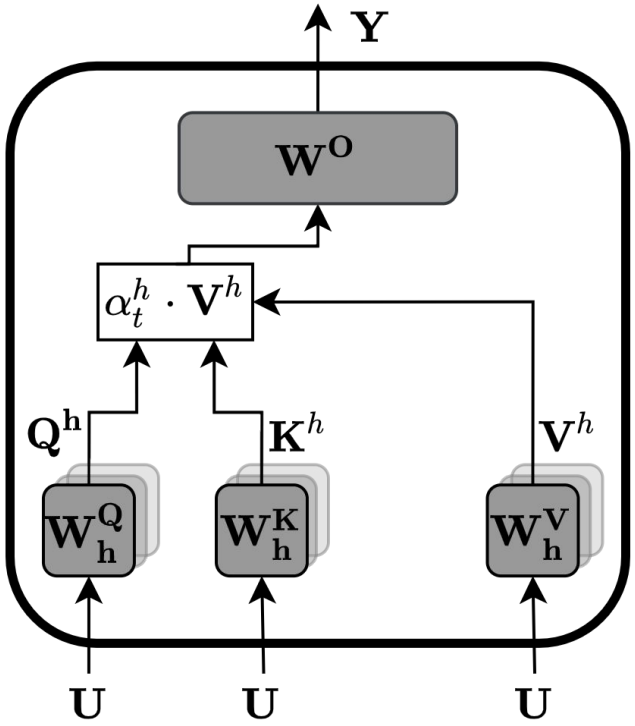
# TransformerLayer




```python
class TransformerLayer(nn.Module):
    def __init__(self, d_model, dim_feedforward=512, dropout=0.1, activation="relu"):
        super().__init__()
        self.self_attn = SelfAttention(d_model)
        # Implementation of Feedforward model
        self.linear1 = nn.Linear(d_model, dim_feedforward)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(dim_feedforward, d_model)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout1 = nn.Dropout(dropout)
        self.dropout2 = nn.Dropout(dropout)

    def forward(self, src):
        src2 = self.self_attn(src)
        src = src + self.dropout1(src2)
        src = self.norm1(src)
        src2 = self.linear2(self.dropout(F.relu(self.linear1(src))))
        src = src + self.dropout2(src2)
        src = self.norm2(src)
        return src
```

# Self-attention




```python
class SelfAttention(nn.Module):
    def __init__(self, embed_dim, bias=True):
        super().__init__()
        self.k_proj = nn.Linear(embed_dim, embed_dim, bias=bias)
        self.v_proj = nn.Linear(embed_dim, embed_dim, bias=bias)
        self.q_proj = nn.Linear(embed_dim, embed_dim, bias=bias)
        self.out_proj = nn.Linear(embed_dim, embed_dim, bias=bias)
        self.reset_parameters()

    def reset_parameters(self):
        # Empirically observed the convergence to be much better with the scaled initialization
        nn.init.xavier_uniform_(self.k_proj.weight, gain=1 / math.sqrt(2))
        nn.init.xavier_uniform_(self.v_proj.weight, gain=1 / math.sqrt(2))
        nn.init.xavier_uniform_(self.q_proj.weight, gain=1 / math.sqrt(2))
        nn.init.xavier_uniform_(self.out_proj.weight)
        if self.out_proj.bias is not None:
            nn.init.constant_(self.out_proj.bias, 0.)

    # B = Batch size
    # W = Number of context words (left + right)
    # E = embedding_dim
    def forward(self, x):
        # x shape is (B, W, E)
        q = self.q_proj(x)
        # q shape is (B, W, E)
        k = self.k_proj(x)
        # k shape is (B, W, E)
        v = self.v_proj(x)
        # k shape is (B, W, E)
        y, _ = attention(q, k, v)
        # y shape is (B, W, E)
        y = self.out_proj(y)
        # y shape is (B, W, E)
        return y
```

# Scaled Dot Product Attention

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
             / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -Inf)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

# Task : Improve the Transformer Baseline

Suggestions:

- Try a Feedforward Neural Network Language Model
- Increase the number of TransformerLayers (2 o more)
- TransformerLayer with multi-head attention
- Hyperparameter optimization: embedding size, batch size, pooling layer (mean, max, first, ...), optimizer, learning rate/scheduler, number of epochs, etc.