RNN from scratch

Muhammad Tamjid Rahman

Contents

Loading R packages

```
library(uuml)
```

1 Transformers and Attention

Loading dataset

```
data("transformer_example")
```

1)

```
# Picking out the matrix for the first attention head
Wq <- transformer_example$Wq[,,1]
Wk <- transformer_example$Wk[,,1]
Wv <- transformer_example$Wv[,,1]
```

```
# Picking out the first three words and their embeddings
X <- transformer_example$embeddings[1:3,]
```

Implementing the function qkv()

```
qvk = function(X, Wq, Wk, Wv){
    Q=X%*%Wq
    K=X%*%Wk
    V=X%*%Wv
    return(list(Q=Q,K=K,V=V))
}
```

```
qvk(X,Wq,Wk,Wv)
```

```
## $Q
##              [,1]        [,2]       [,3]
## the    0.4722259  0.04995783 -0.5350845
## quick -0.3662435  0.12144160  0.3454785
## brown -0.1029677 -0.12728414  0.1817097
##
## $K
##                [,1]         [,2]       [,3]
## the    0.094360579 -0.203807092 -0.1851229
## quick -0.033313240  0.279012100  0.2530560
## brown -0.004457052  0.001013468  0.0133802
##
## $V
##                [,1]        [,2]        [,3]
## the    0.317318525 -0.35023010  0.13284078
## quick  0.009929565  0.04208206 -0.15412097
## brown -0.316413241  0.27717408  0.02725089
```

2 Computing the attention

```
res=qvk(X,Wq,Wk,Wv)

# Softmax activation function

softmax <- function(qk){
    val <-exp(qk)/sum(exp(qk))

    return(val)
}
```

```
attention = function(Q,K,V){
    attention = matrix(0,nrow(K),nrow(K))
    for (i in 1:nrow(K)) {
        attention[i,] = softmax((Q%*%t(K))[i,]/sqrt(nrow(K)))
    }
    Z = attention%*%V
    return(list(Z = Z, attention = attention))
}

attention(res$Q, res$K, res$V)
```

```
## $Z
##               [,1]          [,2]         [,3]
## [1,]  0.012395453 -0.0212420459  0.009404870
## [2,] -0.003759269 -0.0008360029 -0.005108890
## [3,]  0.002412222 -0.0088974612  0.001147999
##
## $attention
##           [,1]      [,2]      [,3]
## [1,] 0.3601932 0.3080896 0.3317172
## [2,] 0.3088780 0.3582373 0.3328847
## [3,] 0.3300375 0.3360583 0.3339042
```

3 Interpreting

Attention values describes how much the attention of a word to the other word in a sequence. In the second row, the work 'quick' has 31% attention to the word 'the', 36% at its own and 33% to 'brown'.

4 Implementing a multi-head attention layer

```
multi_head_self_attention = function(X,Wq,Wk,Wv,W0){
    Z = matrix(0,ncol(Wk),1)

    for (i in 1:dim(Wk)[3]) {
        Wq1 <- Wq[,,i]
        Wk1 <- Wk[,,i]
        Wv1 <- Wv[,,i]

        Q = qvk(X,Wq1,Wk1,Wv1)$Q
        V = qvk(X,Wq1,Wk1,Wv1)$V
        K = qvk(X,Wq1,Wk1,Wv1)$K

        Z = cbind(Z,attention(Q,K,V)$Z)
    }
    return(Z[,-1]%*%W0)
}

multi_head_self_attention(X,
                          transformer_example$Wq,
                          transformer_example$Wk,
                          transformer_example$Wv,
                          transformer_example$W0)
```

```
##              [,1]          [,2]         [,3]
## [1,] -0.014189613 -0.0040299008 -0.006756286
## [2,] -0.009963516 -0.0010724342 -0.001996524
## [3,] -0.006394562 -0.0006626115 -0.002219108
```

2 Implementing a simple RNN

Loading Data

```
data("rnn_example")
```

1

```
hidden_dim=4
output_dim=3

rnn_unit=function(h_t_minus_one,X,W,U,b){
  a_t=b+W%*%h_t_minus_one+U%*%t(X)
  h_t_minus_one=tanh(a_t)
  return(a_t=a_t)
}
```

```r
X <- rnn_example$embeddings[1,,drop=FALSE]
h_t_minus_one <- matrix(0, nrow = hidden_dim, ncol = 1)
a_t <- rnn_unit(h_t_minus_one, X,
                W = rnn_example$W,
                U = rnn_example$U,
                b = rnn_example$b)

a_t
```

```
##              the
## [1,]  0.5819145
## [2,] -2.2686535
## [3,] -0.6410312
## [4,]  1.4891931
```

2 Implementing the tanh() activation function

```r
activation=function(a_t){
  h_t=tanh(a_t)
  return(h_t=h_t)
}

h_t <- activation(a_t)
h_t
```

```
##              the
## [1,]  0.5240555
## [2,] -0.9788223
## [3,] -0.5656013
## [4,]  0.9031762
```

3 Implementing the output function and the softmax function

```r
output_rnn=function(h_t,V,c){
  o_t= c+ V%*%h_t
  return(output_rnn=o_t)
}

yhat_t <- softmax(output_rnn(h_t, rnn_example$V, rnn_example$c))
yhat_t
```

```
##             the
## [1,] 0.3063613
## [2,] 0.2930885
## [3,] 0.4005502
```

4 Implementing the full recurrent layer

```r
rnn_layer=function(X,W,V,U,b,c){

  h_t_minus_one <- matrix(0, nrow = hidden_dim, ncol = 1)
  h_t=matrix(0, nrow = hidden_dim,1)
  yhat=matrix(0,nrow = output_dim,1)
```

```
  for (i in 1:nrow(X)) {
    a_t <- rnn_unit(h_t_minus_one, t(X[i,]),W,U,b)

    o_t=output_rnn(activation(a_t), V, c)

    h_t=cbind(h_t,activation(a_t))

    yhat=cbind(yhat, softmax(o_t))

    h_t_minus_one=tanh(a_t)

  }

  return(list(h_t=t(h_t[,-1]),yhat=t(yhat[,-1])))
}

X <- rnn_example$embeddings[1:3,,drop=FALSE]
rnn_layer(X,
          W = rnn_example$W,
          V = rnn_example$V,
          U = rnn_example$U,
          rnn_example$b,
          rnn_example$c)
```

```
## $h_t
##             [,1]        [,2]       [,3]       [,4]
## [1,]  0.52405551 -0.97882227 -0.5656013  0.9031762
## [2,] -0.05951368  0.03988226  0.8241800 -0.6562744
## [3,] -0.08984008  0.92822217 -0.1563247 -0.6657626
##
## $yhat
##           [,1]      [,2]      [,3]
## [1,] 0.3063613 0.2930885 0.4005502
## [2,] 0.2838013 0.3490452 0.3671536
## [3,] 0.2878002 0.3666877 0.3455121
```

5 The hidden state h_t for the token dog

```
X <- rnn_example$embeddings[drop=FALSE]

h_t_dog=rnn_layer(X,
          W = rnn_example$W,
          V = rnn_example$V,
          U = rnn_example$U,
          rnn_example$b,
          rnn_example$c)$h_t[9,]



h_t_dog
```

```
## [1] -0.2928465  0.1813845 -0.2190118  0.2592397
```

2 Implementing a simple RNN

Loading Data

```
data("rnn_example")
```

1

```
hidden_dim=4
output_dim=3

rnn_unit=function(h_t_minus_one,X,W,U,b){
  a_t=b+W%*%h_t_minus_one+U%*%t(X)
  h_t_minus_one=tanh(a_t)
  return(a_t=a_t)
}


X <- rnn_example$embeddings[1,,drop=FALSE]
h_t_minus_one <- matrix(0, nrow = hidden_dim, ncol = 1)
a_t <- rnn_unit(h_t_minus_one, X,
                W = rnn_example$W,
                U = rnn_example$U,
                b = rnn_example$b)

a_t
```

```
##             the
## [1,]  0.5819145
## [2,] -2.2686535
## [3,] -0.6410312
## [4,]  1.4891931
```

2 Implementing the tanh() activation function

```
activation=function(a_t){
  h_t=tanh(a_t)
  return(h_t=h_t)
}

h_t <- activation(a_t)
h_t
```

```
##             the
## [1,]  0.5240555
## [2,] -0.9788223
## [3,] -0.5656013
## [4,]  0.9031762
```

3 Implementing the output function and the softmax function

```
output_rnn=function(h_t,V,c){
  o_t= c+ V%*%h_t
```

```
  return(output_rnn=o_t)
}

yhat_t <- softmax(output_rnn(h_t, rnn_example$V, rnn_example$c))
yhat_t
```

```
##            the
## [1,] 0.3063613
## [2,] 0.2930885
## [3,] 0.4005502
```

4 Implementing the full recurrent layer

```
rnn_layer=function(X,W,V,U,b,c){

  h_t_minus_one <- matrix(0, nrow = hidden_dim, ncol = 1)
  h_t=matrix(0, nrow = hidden_dim,1)
  yhat=matrix(0,nrow = output_dim,1)

  for (i in 1:nrow(X)) {
    a_t <- rnn_unit(h_t_minus_one, t(X[i,]),W,U,b)

    o_t=output_rnn(activation(a_t), V, c)

    h_t=cbind(h_t,activation(a_t))

    yhat=cbind(yhat, softmax(o_t))

    h_t_minus_one=tanh(a_t)

  }

  return(list(h_t=t(h_t[,-1]),yhat=t(yhat[,-1])))
}

X <- rnn_example$embeddings[1:3,,drop=FALSE]
rnn_layer(X,
          W = rnn_example$W,
          V = rnn_example$V,
          U = rnn_example$U,
          rnn_example$b,
          rnn_example$c)
```

```
## $h_t
##             [,1]        [,2]       [,3]       [,4]
## [1,]  0.52405551 -0.97882227 -0.5656013  0.9031762
## [2,] -0.05951368  0.03988226  0.8241800 -0.6562744
## [3,] -0.08984008  0.92822217 -0.1563247 -0.6657626
##
## $yhat
##           [,1]      [,2]      [,3]
## [1,] 0.3063613 0.2930885 0.4005502
## [2,] 0.2838013 0.3490452 0.3671536
## [3,] 0.2878002 0.3666877 0.3455121
```

5 The hidden state h_t for the token dog

```
X <- rnn_example$embeddings[drop=FALSE]

h_t_dog=rnn_layer(X,
          W = rnn_example$W,
          V = rnn_example$V,
          U = rnn_example$U,
          rnn_example$b,
          rnn_example$c)$h_t[9,]



h_t_dog
```

```
## [1] -0.2928465  0.1813845 -0.2190118  0.2592397
```

The role of the parameters U, W in the recurrent net

U is the weight matrix for the input vector x and W is the weight matrix for the state of previous time step in hidden layer (eq 10.8).[1] In our model in simple_rnn (SimpleRNN) layer the dimension of W is 32x16 and for U is 16x16.

The role of the parameters V

The parameter V is the weight matrix for the hidden-to-output connections (eq. 10.10.[1] In our model in dense_2 (Dense) layer it is included. The dimension of V is 32x32.