Char Level CNN-LM

Piya Amara Palamure

03-31-2024

How the char level prediction are done using NN like CNN architecture

- In the previous use of MLP-LM, we employed a context length of three characters alongside embedding vector of size 10 to forecast the succeeding character.
- Before inputting the embedded three characters into the MLP-NN, we concatenated the three-character vectors into a single large vector.
- However, when the embedded vector is big and the context length is large, it presents an overloaded information for the NN to manage.
- To address this challenge, instead of concatenating the entire embedded context vector, we propose subdividing it into smaller chunks and concatenating each chunk before feeding it into NN.
- This hierarchical implementation of deep NN facilitates efficient data handling and enhance prediction accuracy.

Context length = 3

```
... --> e
..e --> m
.em --> m
emm --> a
mma --> .
... --> o
..o --> l
.ol --> i
oli --> v
liv --> i
ivi --> a
via --> .
... --> a
..a --> v
.av --> a
ava --> .
... --> i
..i --> s
.is --> a
isa --> b
```

Context length = 8

```
........ --> e
.......e --> m
......em --> m
.....emm --> a
....emma --> .
........ --> o
.......o --> l
......ol --> i
.....oli --> v
....oliv --> i
...olivi --> a
..olivia --> .
........ --> a
.......a --> v
......av --> a
.....ava --> .
........ --> i
.......i --> s
......is --> a
.....isa --> b
```

- Word list

```
Words[:10] = ['emma’, 'olivia’, 'ava’, 'isabella’, 'sophia’, 'charlotte’, 'mia’,
'amelia’, 'harper’, 'evelyn']
```

- Char to integer encoding

Encode = `{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26, '.': 0}`

…….	--> e
……e	--> m
……em	--> m
……emm	--> a
….emma	--> .
…….	--> o
……o	--> l
……ol	--> i
……oli	--> v
….oliv	--> i
…olivi	--> a
..olivia	--> .
…….	--> a
……a	--> v
……av	--> a
……ava	--> .
…….	--> i
……i	--> s
……is	--> a
……isa	--> b

```
[[ 0, 0, 0, 0, 0, 0, 0, 0],
 [ 0, 0, 0, 0, 0, 0, 0, 5],
 [ 0, 0, 0, 0, 0, 0, 5, 13],
 [ 0, 0, 0, 0, 0, 5, 13, 13],
 [ 0, 0, 0, 0, 5, 13, 13, 1],
 [ 0, 0, 0, 0, 0, 0, 0, 0],
 [ 0, 0, 0, 0, 0, 0, 0, 15],
 [ 0, 0, 0, 0, 0, 0, 15, 12],
 [ 0, 0, 0, 0, 0, 15, 12, 9],
 [ 0, 0, 0, 0, 15, 12, 9, 22],
 [ 0, 0, 0, 15, 12, 9, 22, 9],
 [ 0, 0, 15, 12, 9, 22, 9, 1],
 [ 0, 0, 0, 0, 0, 0, 0, 0],
 [ 0, 0, 0, 0, 0, 0, 0, 1],
 [ 0, 0, 0, 0, 0, 1 0, 1, 22],
 [ 0, 0, 0, 0, 0,, 22, 1],
 [ 0, 0, 0, 0, 0, 0, 0, 0],
 [ 0, 0, 0, 0, 0, 0, 0, 9],
 [ 0, 0, 0, 0, 0, 0, 9, 19],
 [ 0, 0, 0, 0, 0, 9, 19, 1]]
```

```
[ 5,
13,
13,
1,
0,
15,
12,
9,
22,
9,
1,
0,
1,
22,
1,
0,
9,
19,
1,
2])
```

- The shape of the embedding vector is represented as C(27,24), where we have 27 characters, and each character is embedded with a vector of size 24.
- The input to the neural network, assuming an 8-character context length, is initially (8,24). However, as we concatenate 2 characters together, the input size of the network becomes (4,2*24) = (4,48).
- In this setup, the first dimension in the input can be understood as the batch dimension. Therefore, 4 sets of the two concatenated characters (each of size 48) will be processed in parallel, resulting in an output shape of (4, n_hidden).
- After the first hidden layer, 2 batches from the first hidden layer (out of 4) will be concatenated together to form the input of the second hidden layer. Consequently, the size of the input for the second hidden layer is (2, 2*n_hidden), where 2 represents the new batch dimension.
- Finally, the two batches from the second hidden layer will be concatenated and fed into the third hidden layer as input.

```
# hierarchical network
n_embd = 24             # character embedding vectors dim
n_hidden = 128           # the number of neurons in the hidden layer of the MLP
model = Sequential([
    Embedding(vocab_size, n_embd),
    FlattenConsecutive(2), Linear(n_embd * 2, n_hidden, bias=False), BatchNorm1d(n_hidden), Tanh(),
    FlattenConsecutive(2), Linear(n_hidden*2, n_hidden, bias=False), BatchNorm1d(n_hidden), Tanh(),
    FlattenConsecutive(2), Linear(n_hidden*2, n_hidden, bias=False), BatchNorm1d(n_hidden), Tanh(),
    Linear(n_hidden, vocab_size),
])
```

Let's consider a single input example : ...olivi --> a