Char Level MLP-LM

Piya Amara Palamure

03-29-2024

How the char level prediction are done using MLP

- In the previous approach, we utilized a single character to predict the subsequent character. In this iteration, we extend the context length to three characters, aiming to predict the next character based on this longer context.
- Initially, the characters are encoded into integers. Subsequently, each integer, representing a character, undergoes embedding using d-dimensional vectors, such as a 10-dimensional vector.
- During training, we employ a batch size of 32, meaning that weights are updated using 32 input examples simultaneously.
- We initialize both weight matrices and bias matrices with random numbers, and these parameters are adjusted during backpropagation. Additionally, we apply the tanh() non-linearity at the hidden node.
- The output (logits) of the Multilayer Perceptron (MLP) is then fed into a cross-entropy function along with the target integer values to compute the loss. It's important to note that this loss is calculated per mini-batch.

- Word list

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

- Input vector (context vector with 3 chars) list

```
Context_vec[:20] =[‘…’, ‘..e’, ‘.em’, ‘emm’, ‘mma’, ‘…’, ‘..o’, ‘.ol’, ‘oli’,
‘liv’, ‘ivi’, ‘via’, ‘…’, ‘..a’, ‘.av’, ‘ava’, ‘…’, ‘..i’, ‘.is’, ‘isa’]
```

- 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}`

- Input integer list

```
Input_int[:20] = [[ 0, 0, 0], [ 0, 0, 5], [ 0, 5, 13], [ 5, 13, 13], [13, 13, 1], [ 0, 0, 0], [ 0,
0, 15], [ 0, 15, 12], [15, 12, 9], [12, 9, 22], [ 9, 22, 9], [22, 9, 1], [ 0, 0, 0], [ 0, 0, 1], [
0, 1, 22], [ 1, 22, 1], [ 0, 0, 0], [ 0, 0, 9], [ 0, 9, 19], [ 9, 19, 1]]

Output_int[:20]= [ 5, 13, 13, 1, 0, 15, 12, 9, 22, 9, 1, 0, 1, 22, 1, 0, 9, 19, 1, 2]
```

- Shape of embedding vector: C(27,10), Note: we have 27 characters, and each char can be embedded with vector of size 10

- The input to the MLP (assuming 3-character context length) is: (3,10). However, we concatenate 3 characters together. So, the input size of the network is: (30)

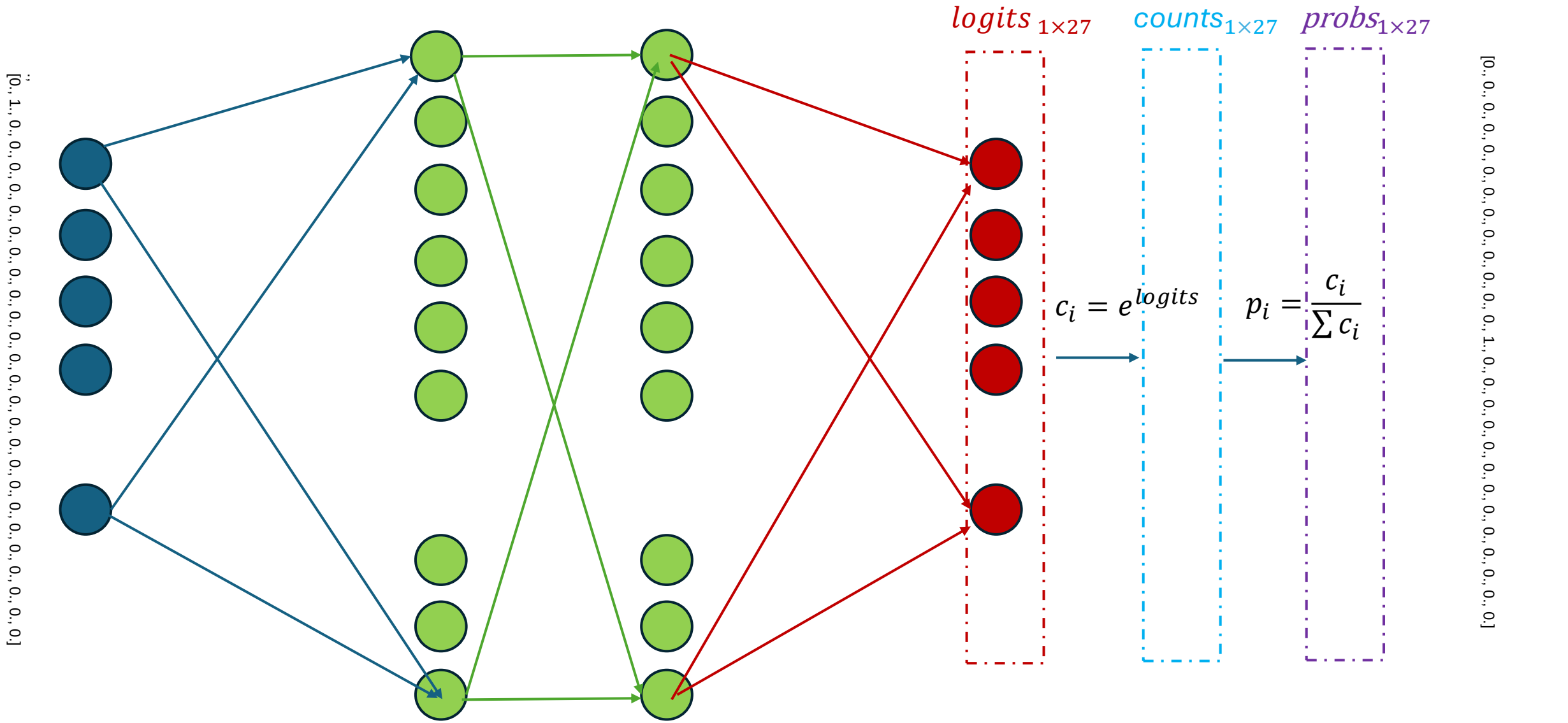
- If the batch size is 32 (i.e. we want to process 32 inputs in parallel to update the weights of NN), then the input embedded matrix can be arranged to have size of: (32,30)

```
X_emb = [ [[ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02],
           [ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02],
           [ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02]],
          [[ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02],
           [ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02],
           [-1.2159e-01, 4.6097e-02, 4.1425e-02, -2.8750e-02, -2.6748e-02, 1.4824e-01, 3.0152e-04, 1.1143e-01, -2.6830e-01, -1.3763e-01]],
          [[ 1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02],
           [-1.2159e-01, 4.6097e-02, 4.1425e-02, -2.8750e-02, -2.6748e-02, 1.4824e-01, 3.0152e-04, 1.1143e-01, -2.6830e-01, -1.3763e-01],
           [-9.9644e-02, -1.3846e-01, 1.3958e-02, -8.6713e-02, -2.6210e-01, 1.6710e-01, 1.8228e-01, -3.4814e-02, 7.7594e-02, 4.3900e-01]],
          [[-1.2159e-01, 4.6097e-02, 4.1425e-02, -2.8750e-02, -2.6748e-02, 1.4824e-01, 3.0152e-04, 1.1143e-01, -2.6830e-01, -1.3763e-01],
           [-9.9644e-02, -1.3846e-01, 1.3958e-02, -8.6713e-02, -2.6210e-01, 1.6710e-01, 1.8228e-01, -3.4814e-02, 7.7594e-02, 4.3900e-01],...
           [-9.9644e-02, -1.3846e-01, 1.3958e-02, -8.6713e-02, -2.6210e-01, 1.6710e-01, 1.8228e-01, -3.4814e-02, 7.7594e-02, 4.3900e-01],
           [ 4.1091e-02, -5.3039e-02, 1.2489e-01, -9.2509e-02, -6.3775e-02, 2.0687e-01, -1.5866e-01, 8.0689e-02, -7.7546e-02, -1.0073e-01]]]
```

```
X_in =  [[1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02,
          1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02,
          1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02]

         [1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02,
          1.6560e-01, -5.3304e-03, 1.0905e-01, -2.8423e-02, 1.4621e-01, -9.7864e-02, 5.7133e-02, -3.1479e-02, -4.3191e-02, 4.4227e-02,
         -1.2159e-01, 4.6097e-02, 4.1425e-02, -2.8750e-02, -2.6748e-02, 1.4824e-01, 3.0152e-04, 1.1143e-01, -2.6830e-01, -1.3763e-01]…..]
```

Ex. One training example (char) is passing through NN



$Input_{1\times30}$

$W_{1(30\times64)}$

$b_{1(1,64)}$

$W_{2(64\times64)}$

$b_{2(1,64)}$

$W_{2(64\times27)}$

$b_{3(1,64)}$

$Target_{1\times27}$

- Forward pass through the NN and Loss function

- $\boldsymbol{logits} = \left(\boldsymbol{Tanh}\left(\left(\boldsymbol{Tanh}(\boldsymbol{X_{in}} \times \boldsymbol{W_1} + \boldsymbol{b_1})\right) \times \boldsymbol{W_2} + \boldsymbol{b_2}\right)\right)\boldsymbol{W_3} + \boldsymbol{b_3}$ [perform this for a single batch]

- $\boldsymbol{loss} = \boldsymbol{CrossEntrophy}(\boldsymbol{logits}, \textbf{integer encoded output})$, this is same as finding counts, get the probabilities finally calculating the negative log-liklihood loss [per batch]

- Forward pass through the NN and Loss function

- Update the weights via backpropergation: $\boldsymbol{w_{lk}}(\boldsymbol{t}+\mathbf{1}) = \mathbf{w_{lk}}(\mathbf{t}) - \boldsymbol{\gamma_t}\frac{\boldsymbol{\partial Loss}}{\boldsymbol{\partial w_{lk}}(\boldsymbol{t})}$ [*the weights are updated at each batch*]

- Note that this model contains significantly more parameters compared to the previous one. Furthermore, we've enhanced it by incorporating contextualized meaning into the input.

- The embedding weights are also fine-tuned during backpropagation. This facilitates a broader phase space for characters to learn, meaning that similar words will cluster closer together.