CSE 151B Deep Learning
UC San Diego, Spring 2022

Homework 2

Name: William Sun UID: A16013590

1 Convolutional Neural Networks

Problem A: Convolution

i.

Solution A.i:

8 filters, each with $5\times5\times3$ *weights plus a bias term per filter is equal to* $(8\times5\times5\times3)+8 = 608$ *parameters.*

ii.

Solution A.ii:

$W_{out} = H_{out} = (W - F + 2P)/S + 1 = (32 - 5 + 0)/1 + 1 = 28$. *Thus, the output tensor has shape* $(28, 28, 8)$ *where* 8 *is based on the number of filters.*

Problem B: Pooling

i.

Solution B.i:

Calculating average using floating point numbers:

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 0.25 \end{bmatrix}$$

$$\begin{bmatrix} 0.5 & 1 \\ 0.25 & 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.5 & 0.25 \\ 1 & 0.5 \end{bmatrix}$$

ii.

Solution B.ii:

All 4 matrices have the same result:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

2 Recurrent Neural Networks

Problem A: LSTM

i.

Solution A.i:

Use cross-entropy loss: $L_t = -y_t \log \hat{y}_t$

where $\hat{y}_t = \sigma(W_y h_t)$

1) $\delta c_t = \frac{\partial L_t}{\partial c_t} = \underbrace{\frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t}}_{(\hat{y}_t - y_t) W_y^T} \cdot \frac{\partial h_t}{\partial c_t}$, $\frac{\partial (o_t \odot \tanh(c_t))}{\partial c_t} = o_t \odot \tanh'(c_t) = o_t (1 - \tanh^2(c_t))$

Thus, $\boxed{\delta c_t = (\hat{y}_t - y_t) W_y^T \cdot o_t (1 - \tanh^2(c_t))}$

2) $\delta o_t = \frac{\partial L_t}{\partial o_t} = \underbrace{\frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t}}_{(\hat{y}_t - y_t) W_y^T} \cdot \underbrace{\frac{\partial h_t}{\partial o_t}}_{\tanh(c_t)}$

Thus, $\boxed{\delta o_t = (\hat{y}_t - y_t) W_y^T \cdot \tanh(c_t)}$

3) $\delta i_t = \frac{\partial L_t}{\partial i_t} = \underbrace{\frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial c_t}}_{\text{from (1)}} \cdot \underbrace{\frac{\partial c_t}{\partial i_t}}_{\tanh(W^{ox} x_t + W^{oh} h_{t-1})}$

Thus, $\boxed{\delta i_t = (\hat{y}_t - y_t) W_y^T \cdot o_t (1 - \tanh^2(c_t)) \cdot \tanh(W^{ox} x_t + W^{oh} h_{t-1})}$

4) $\delta f_t = \frac{\partial L_t}{\partial f_t} = \underbrace{\frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial c_t}}_{\text{from (1)}} \cdot \underbrace{\frac{\partial c_t}{\partial f_t}}_{c_{t-1}}$

Thus, $\boxed{\delta f_t = (\hat{y}_t - y_t) W_y^T \cdot o_t (1 - \tanh^2(c_t)) \cdot c_{t-1}}$

ii.

Solution A.ii:

Based on the derivations above, the gradient does not explode if the forget gates are close to 1 and the input/output gates are close to 0. Looking at the gradient of c_t, if the output gate is close to zero, the gradient approaches 0 too, which means the system remains stable and unchanged. Additionally, since the forget gate is close to 1 and input is close to 0, the cell state will keep its previous value, thereby preventing both an exploding gradient and vanishing gradient. The gradient of h_t will also not explode. This gradient is derived as the left portion of part (1) in solution A.i, and this shows that there is no term which can exponentially explode when the forget gates are close to 1 and the input/output gates are close to 0.

3 Poem Generation

Problem A: Pre-processing

i.

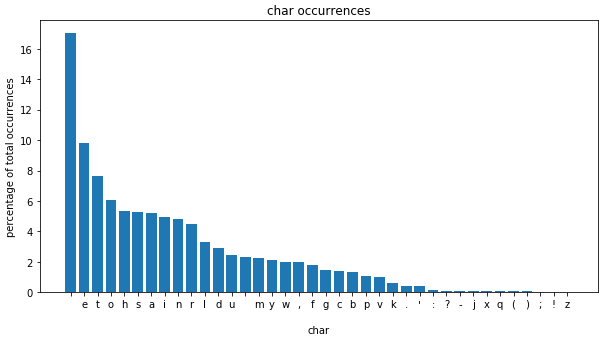
Solution A.i:

My pre-processing steps are all used to make the model training step more straightforward. Since the eventual training will use 40 characters at a time, it will roughly be training on one line of the data at a time. In this sense, I only want the model to train on each sonnet (14 lines of the poem) as the best way to learn word structure and word generation. With 200 LSTM units in the end, there is not enough history to learn the full sonnet structure, so I opted to remove the intermediary lines and numbering between each sonnet, because this would just confuse the model on what to generate. Additionally, to simplify the complexity of the data, I remove capitalization from the characters. Since the eventual training will use character-based LSTM, I tokenize the data into characters, with each sequence being each line in the pre-processed data. Initially I tried tokenizing via words based on d2l's default code, but this would not be helpful in the model training step, so I went to the character based tokenization, and mapped each unique character to a unique integer for the training.

ii.

Solution A.ii:

The statistical analysis done on the pre-processed data is to count how many times each character appears in the dataset. This statistic is useful because we can compare the LSTM results to this distribution to check if the LSTM is working properly. The output distribution of characters should be roughly comparable to the input distribution. For convenience, the data is plotted as a percentage of total characters. The visualization is shown in the bar graph.



Problem B: Model Training

i.

Solution B.i:

The model implemented for the RNN is Andrej Karpathy's minimal character-level vanilla RNN model. This model uses 100 hidden layers as suggested by the write-up. Initially, 200 hidden layers were used, but as this was tuned, the loss/result was clearly better with 100 hidden layers. Overall, 200, 150, and 100 were the hyperparameters tuned for the size of the network. The sequence length was kept at 40 as suggested by the write-up, and the learning rate was kept the same as well. The model itself uses adagrad for optimization as it is much more effective than SGD.

Below are the generated poems at the end of training. The model did a good job of picking up common words such as "thou, your, self, lover, thy, dead, fall." This is a good sign because they are found throughout Shakespeare's poetry, and the same word structure, spelling, and placement is found in the generated poems. Additionally, given that each line is around 40 characters, we expect the resulting poem with 200 characters to be around 5 lines, and the model does a good job picking this up with its placement of newlines. Finally, the poems have a pattern of having two spaces in front of some lines, and the model also picks up on this, where it occasionally places two spaces in front of newlines.

ii.

Solution B.ii:

As explained in part i and looking at the above examples, the RNN does a good job of learning sentence structure, but not necessarily sonnet structure. Firstly, as explained in part i, given 200 characters, we expect there to be around 5 lines, and this pattern does show up across the board in the generated poems. This means it does a good job with sentence structure in terms of placement of newlines. The placement of spaces in the sentence structure is also spot on, where all the words understandable by humans have a space after them. Also as mentioned previously, the model occasionally places two spaces in front of newlines, which follows the input data. However, given 200 characters, the model cannot learn a full 14 line sonnet, so it does not successfully learn sonnet structure. The runtime for the RNN was with 30000 iterations in each training session, and the amount of data provided was the entirety of the pre-processed Shakespeare data.

iii.

Solution B.iii:

The generated poems of 1.5, 0.75, and 0.25 are shown above. There is a significant difference between the results given the 40 character seed. With a high temperature, the model becomes less confident about its prediction, hence becoming "softer." The lower the temperature, the model becomes more confident about its prediction, hence becoming "harder." This qualitative description is reflected in the output. When the temperature is high (1.5), the model is less sure of what character it should pick, so there is a more diverse output of what character is selected, hence having more rare characters output like '-' and more newlines. When the temperature is lower (0.25), the model is more confident, so it is more likely to select tokens that are more common, and hence the output creates very common patterns with words such as "thou, the, so, here." A temperature of (0.75) is closest to normal (1), so that output is also the most similar to our original training results, although since it is slightly more confident that (1), we see more common words like "thou" and "thee".