Conformers Homework

Kevin Wang, Timber Lin, Theophilus Pedapolu

April 2023

In this homework, we guide you through the Conformer, a novel deep-learning architecture that combines convolutional layers with transformers to capture both the local and global dependencies of an audio sequence. It has shown state-of-the-art results on automated speech recognition (ASR) tasks. The first 2 questions are analytical questions meant to help you understand a new convolution operation and activation function you may not have heard of but which are key components of the Conformer. The rest of the homework deals with implementing and training a conformer. The original Conformer paper is here: https://arxiv.org/abs/2005.08100 We encourage you to read through this paper before starting this homework.

1 Depthwise-Separable Convolutions

So far, we have learned about the convolution operation in which we take a kernel and slide it over an input, multiplying element-wise and summing at each position to produce an output. Although convolutional layers require far fewer computations than FFN layers, there are cases when we would like to reduce the computation even further. This is the motivation behind depthwise-separable convolutions. These are a type of convolution that "separate" the standard convolution procedure into two parts: depthwise convolution and pointwise convolution.

Suppose we have a $M \times N \times D$ input tensor. In standard convolution, we would use D kernels of size $K \times K \times D$ where K is the kernel size and C is the number of desired output channels. The output would then be of the form $M' \times N' \times C$ Notice that the depth of the kernel equals the depth of the input tensor. In depthwise convolution, however, we separate the depth dimension and instead have D kernels of size $K \times K \times 1$. Each kernel is convolved separately at a different depth to produce outputs of shape $M' \times N' \times 1$. These outputs are then stacked to produce a tensor of shape $M' \times N' \times D$. Figure 1 visually shows this procedure.

Pointwise convolution is then applied on the $M' \times N' \times D$ tensor to change D to a desired number of channels C. The term "pointwise" refers to the fact that this type of convolution uses C kernels of shape $1 \times 1 \times D$. Each kernel is applied on the entire tensor to produce an output of shape $M' \times N' \times 1$. These outputs are again stacked in the channel dimension to produce a final $M' \times N' \times C$ tensor, the same shape we would get in standard convolution. As you can see depthwise-separable convolutions are very similar to standard convolutions, except that we selectively apply the kernels first to the 2D dimensions (M and N) then across the channel dimension (D) instead of all dimensions at once.

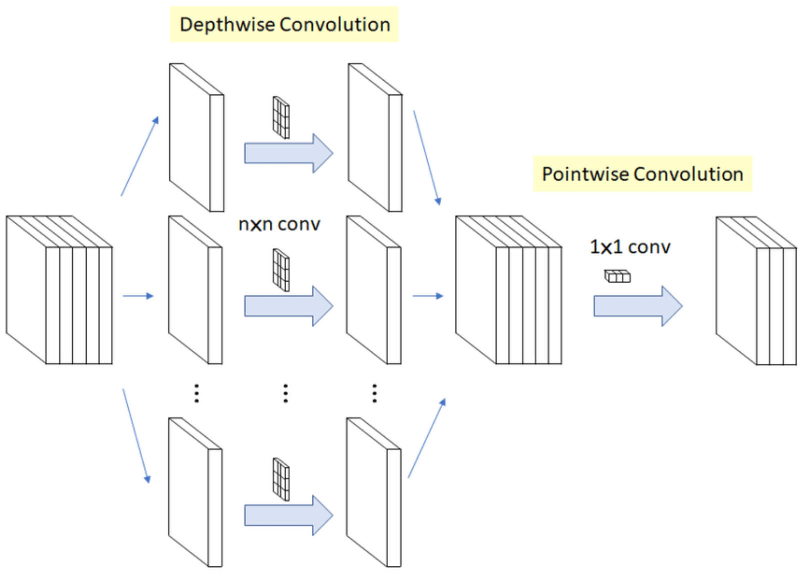

Figure 1: Visual of how depthwise and pointwise convolutions are performed. Source: Analytics Vidhya

Suppose we have the $3 \times 3 \times 3$ input tensor A below

$$A = \begin{bmatrix} \begin{bmatrix} 4 & 8 & 2 \\ 1 & 5 & 9 \\ 7 & 6 & 3 \end{bmatrix} & \begin{bmatrix} 0 & 4 & 6 \\ 9 & 7 & 5 \\ 3 & 8 & 1 \end{bmatrix} & \begin{bmatrix} 7 & 4 & 2 \\ 1 & 0 & 5 \\ 0 & 5 & 1 \end{bmatrix} \end{bmatrix}$$

For subparts to this question, assume all convolutions are performed with stride = 1 and padding = 0

(a) Compute the depthwise convolution on A using the $2 \times 2 \times 1$ kernels below in order

$$K_1 = \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \end{bmatrix} \quad K_2 = \begin{bmatrix} \begin{bmatrix} 4 & 5 \\ 0 & 9 \end{bmatrix} \end{bmatrix} \quad K_3 = \begin{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 8 \end{bmatrix} \end{bmatrix}$$

(b) Perform pointwise convolution on the output you computed in part (a) using the $2 \times 2 \times 3$ kernels below in order. You should have 2 channels in the final output

$$K_1 = [[1] \quad [0] \quad [2]]$$

$$K_2 = [[4] \quad [1] \quad [3]]$$

(c) In this example with the $3 \times 3 \times 3$ input tensor, how many different parameters were used for both the depthwise and pointwise convolution steps? Suppose we instead performed standard convolution with 2 kernels of shape $2 \times 2 \times 3$ to get the same final output shape. How many parameters would we use in this case?

(d) Now we will use the number of multiplications performed as a proxy for computational cost. Again For the depthwise-separable convolution (depthwise and pointwise steps), how many total multiplications were performed. Ignore any addition operations involved in calculating sums. For standard convolution using the same kernels in the previous part, how many multiplications were performed?

(e) Consider the general case, where we have a $M \times N \times D$ input tensor. Suppose we use $K \times K \times 1$ kernels for depthwise convolution and C kernels of size $1 \times 1 \times D$ kernels for pointwise convolution. How many parameters are used in the depthwise-separable convolutional layer altogether? How many multiplications are performed? Calculate the parameters and multiplications for a standard convolutional layer as well assuming we use C kernels of size $K \times K \times D$

(f) Using this information about the number of parameters and the number of multiplications performed, what can you say about the computational and memory efficiency of depthwise-separable convolutions vs. standard convolutions? Is one more efficient than the other?

(g) What are the benefits and drawbacks of depthwise-separable convolutional layers? Why don't we use them all the time?

2 Understanding the Swish Activation Function

The swish activation function is $y = \frac{x}{1+e^{-x}}$, and is sometimes called the sigmoid weighted linear unit. Some of its properties include: being bounded below but not above, non-monotonicity, and smoothness, and these in combination are believed to be advantageous. Although the swish function is inspired by the sigmoid activation function, the swish function critically avoids the vanishing gradient problem suffered by the sigmoid function. Meanwhile, the swish function is quite similar to the relu activation function but importantly has a smooth output which is likely less sensitive to initialization and learning rates, making it easier to optimize than with relu. The swish activation hopes to capture the advantages of both the sigmoid and relu activation functions and has seen better results in deeper networks.

(a) Show that the derivative of the swish activation function can be expressed as $y' = y + \sigma(x) * (1 - y)$ where $\sigma(x) = \frac{1}{1+e^{-x}}$, and explain how this addresses the vanishing gradient problem that affects the sigmoid activation.

(b) How is the swish activation function similar to the self-gating component of the LSTM cell?

(c) There is a modified version of the swish activation function, which includes a tunable β parameter: $z = \frac{x}{1+e^{-\beta x}}$. Show that the derivative of this modified swish activation function can be expressed as $z' = \sigma(\beta x) + \beta x \sigma(\beta x) * (1 - \sigma(\beta x)) = \beta z + \sigma(\beta x) * (1 - z)$

(d) Usually, the value of β is near 1. If we were to increase β towards ∞, what activation function would the modified swish activation approach?

(e) Why would a β value near 0 create issues?

3 Coding Question: Designing the Conformer Architecture

Follow the instructions in this notebook and complete all the TODOs to build the different parts of the conformer and write the pipeline. Make sure you pass all the test cases: Conformers.ipynb

4 Training Conformers for ASR and Ablation Studies

Run through the CoLab notebook Conformers_ASR.ipynb and follow all the steps to train the Conformer models and plot losses. Note that some of the training may take up to 10-15 minutes and please make sure to use a GPU in your runtime. Include your answers to the written questions below

(a)

(i) Experiment with the parameters used to create the Conformer encoder: encoder_dim, num_encoder_layers, depthwise_conv_kernel_size. In general, does increasing or decreasing any of the parameters give better performance?

(ii) Notice that the model takes in an argument called "input_lengths," which contains the lengths of the non-padded sequences for elements in the batch. Why does the conformer encoder need this information? Which part of the pipeline is this most important for?

(iii) What does the decoder consist of for this model? What is contained in the final outputs of the model?

(b)

(i) How does increasing the number of heads in the conformer encoder influence the performance, i.e. the loss rates? Can you give an intuitive explanation as to why?

(ii) In language prediction, multiple heads capture different dependencies in the input text such as subject-verb relationships and pronoun references. What is the audio analog of this idea? In other words, what kind of information do the different heads capture about the audio signal?

(c)

(i) Did one model perform better than the other, either the one with the convolution module first or the one with the self-attention module first?

(ii) In the original paper, the authors also tried to split the input into parallel branches of a multi-headed self attention module and a convolution module with their output concatenated, which yielded worse performance. What does this tell you about the best architectural design for the Conformer block?

5 Homework Process and Study Group

(a) What sources (if any) did you use as you worked through the homework?

(b) If you worked with someone on this homework, who did you work with? List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.

6 References

[1] Gulati, Anmol, et al. "Conformer: Convolution-augmented transformer for speech recognition." arXiv preprint arXiv:2005.08100 (2020).

[2] Ramachandran, Prajit, Barret Zoph, and Quoc V. Le. "Searching for activation functions." arXiv preprint arXiv:1710.05941 (2017)

[3] Kaiser, Lukasz, Aidan N. Gomez, and Francois Chollet. "Depthwise separable convolutions for neural machine translation." arXiv preprint arXiv:1706.03059 (2017).