Conformers Homework Solutions

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.

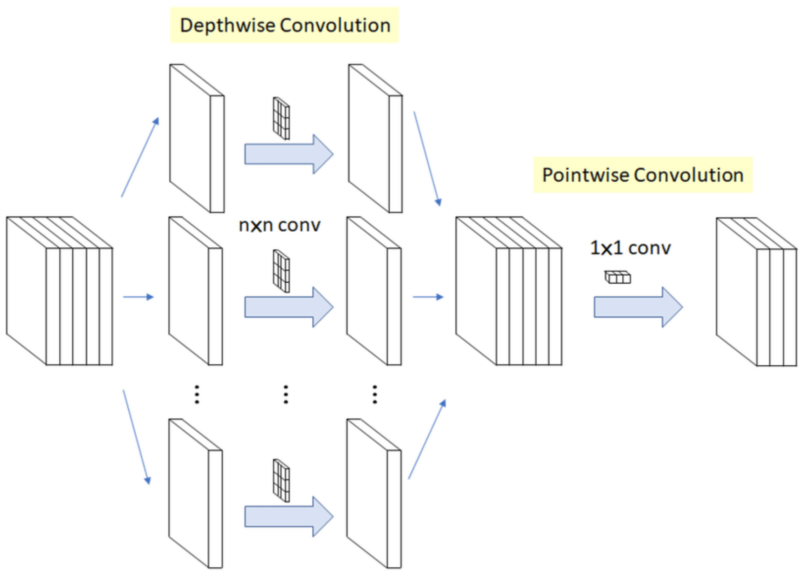

Figure 1: Visual of how depthwise and pointwise convolutions are performed. Source: Analytics Vidhya

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

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

To perform depthwise convolution, we apply K_1, K_2, and K_3 respectively on the 3x3 matrices in each of the channels of A. This gives us the 3 2x2 output matrices:

$$C_1 = \begin{bmatrix} 25 & 39 \\ 37 & 30 \end{bmatrix} \quad C_2 = \begin{bmatrix} 83 & 91 \\ 143 & 62 \end{bmatrix} \quad C_3 = \begin{bmatrix} 7 & 56 \\ 44 & 8 \end{bmatrix}$$

We then stack C_1, C_2, and C_3 along the channel dimension to get the output tensor:

$$Output = \begin{bmatrix} \begin{bmatrix} 25 & 39 \\ 37 & 30 \end{bmatrix} & \begin{bmatrix} 83 & 91 \\ 143 & 62 \end{bmatrix} & \begin{bmatrix} 7 & 56 \\ 44 & 8 \end{bmatrix} \end{bmatrix}$$

(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

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

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

To perform pointwise convolution, we apply K_1 and K_2 across all the depths in the output tensor from part (a). This gets us two 2x2 output channels which we stack together to get the final output tensor:

$$Output = \begin{bmatrix} \begin{bmatrix} 39 & 151 \\ 125 & 46 \end{bmatrix} & \begin{bmatrix} 204 & 415 \\ 423 & 206 \end{bmatrix} \end{bmatrix}$$

(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?

In this example, we used $2 \times 2 \times 3 = 12$ parameters for depthwise convolution and 6 parameters for pointwise convolution for a total of 18 parameters. Standard convolution would use $2 \times 2 \times 3 \times 2 = 24$ parameters

(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?

For depthwise convolution, we performed 4×4 multiplications with each kernel for a total of $4 \times 4 \times 3 = 48$ multiplications. Similarly, for the pointwise layer, we performed $3 \times 4 \times 3 = 36$ multiplication. In total both steps do 84 multiplications. Meanwhile, standard convolution would do $4 \times 3 \times 4 \times 2 = 96$ multiplications

(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 $K \times K \times D$

There are K^2D parameters in the depthwise layer and CD parameters in the pointwise layer for a total of $K^2D + CD$ parameters for depthwise-separable convolution. Meanwhile, standard convolution has K^2CD parameters.In addition, there are $K^2(M-K+1)(N-K+1)D+(M-K+1)(N-K+1)CD = (M-K+1)(N-K+1)(K^2D+CD)$ multiplications performed in depthwise-separable convolution and $(M-K+1)(N-K+1)K^2CD$ multiplications in standard convolution. Notice that the number of multiplications is just the number of parameters times the size of the output along the first 2 dimensions

(f) Using the information in the previous part 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?

In general, depthwise-separable convolution uses fewer parameters and fewer multiplication operations than standard convolution, especially when $C > D$, i.e. we upsample the data. There are cases when standard convolution has less parameters and multiplications (for example when $D >> C$ but the fact that depthwise-separable convolution "splits" up the K^2CD term into $K^2 + CD$ often mean it has better efficiency

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

Depthwise-separable convolutional layers use less space and computational power than standard convolutional layers. The fewer number of parameters may also help to reduce overfitting and increase representational efficiency, which makes them a popular choice for mobile vision applications (read this paper if you want to learn more: `https://arxiv.org/abs/1706.03059`). Although depthwise-separable convolutional layers are more efficient the drawback is that certain models may actually underfit and/or not capture all the information in a dataset due to the lack of parameters. In the Conformer model however, which you will see later, depthwise-separable convolutions work well enough to capture the local relationships in a dataset

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.

$\frac{dy}{dx} = x'\sigma(x) + x\sigma'(x) = \sigma(x) + x\sigma(x)(1-\sigma(x)) = x\sigma(x) + \sigma(x)(1 - x\sigma(x)) = y + \sigma(x)(1-y)$
Unlike the sigmoid, the swish activation's derivative is not bounded by $\frac{1}{4}$ and is close to 1 for larger x values.

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

The swish activation is multiplying x with the sigmoid, which resembles a self-gate, passing x as the sigmoid approaches 1 and passing nothing as the sigmoid approaches 0. Similarly, one of the four LSTM gates is the self-modulation gate. Although the nonlinearity here is $\tanh(x)$, this gate serves a similar purpose by modulating the information from the input gate. As we can see from part (a), for small values of $\sigma(x)$, the derivative is close to the swish output y, and for large values of $\sigma(x)$, the derivative is overwritten to be close to 1.

(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 $z' = \sigma(\beta x) + \beta x\sigma(\beta x)(1-\sigma(\beta x)) = \beta z + \sigma(\beta x)(1-\beta z)$

$\frac{dz}{dx} = x'\sigma(\beta x) + x\sigma'(\beta x) = \sigma(\beta x) + x\beta\sigma(\beta x)(1-\sigma(\beta x)) = x\beta\sigma(\beta x) + \sigma(\beta x)(1 - x\beta\sigma(\beta x)) = \beta z + \sigma(\beta x)(1-\beta z)$

(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?

The modified swish activation approaches the ReLU activation function.

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

The modified swish activation would approach a linear layer, thus losing its desirable nonlinear properties.

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

Solutions: Conformers Sol.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?

There could be different answers to this question but, in general, students should see that increasing all 3 parameters up to a certain point converges to lower losses but increasing them past this point causes overfitting on the data and the model doesn't perform as well

(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?

Recall the code for the Seq2Seq Transformer encoder-decoder architecture. We pad the inputs to make all the sequences the same length but we pass in a padding mask to allow for causal attention masking. This is the same idea here. After transforming the data into a Mel Spectrogram, we have some sequence length which is padded and an input dimension. We want to ensure the padded sequences are not attended to, so we pass in the original lengths to the Conformer encoder. The model transforms these lengths into a mask to be used in the multi-headed self-attention layers. Hence, input_lengths is most important for the transformer module section

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

The decoder for this model is a single-layer LSTM, which is the same decoder used in the Conformers paper. The final output of the model is a tensor of shape (batch_size, num_classes) which contains the log probabilities for classes in each batch.

(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?

Increasing the number of heads generally gives better performance, hence lower loss rates. This is possibly because having more heads allows the model to capture different kinds of long-range dependencies in the data giving it a more complete picture.

(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?

The different heads could capture relationships between frequencies, pitch, amplitude, etc. in the audio signal. Because we transformed the data into a Mel Spectrogram, the model is learning on a frequency scale that is perceptible to humans so it understands these relationships with respect to the human ear

(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?

Students should have achieved lower loss rates for the model with the attention module first but it is quite possible they didn't notice a significant difference because the training set is too small to see the difference between these orderings. They should have consulted the figure in the notebook in this case to determine that having the attention module first performs better

(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?

This tells us that the Conformer block performs slightly better with the self-attention module first then the convolution module instead of some interleaving between the two. Allowing the module to capture long-range dependencies then local features is the best way to go instead of trying to do some form of both at the same time

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).