Topic 6: Convolutional neural networks

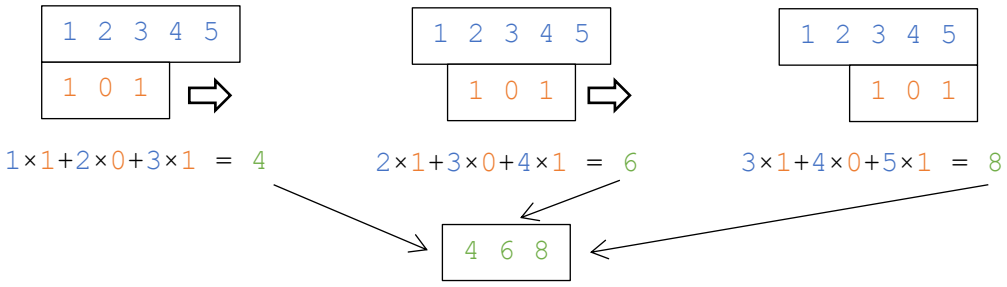
A fully connected neural network (all the neural networks seen till now) can learn to do anything but also has many parameters. More parameters requires more memory, takes longer to train, and requires more training data to learn a generalised function (since more parameters makes it easier to overfit). A fully connected neural network is also the most general way to process data, which means that you're not helping the neural network by limiting it to only do what is likely to work.

One way to limit how a neural network can process input structures is to use convolutional layers, which are layers that are inspired by the way the eye works. Rather than process the whole input at once, you make the neural network take in chunks of the input and process them separately, then combine the results afterwards. This is similar to how you read the text bit by bit rather than all at once.

1. Convolutional layers 1D

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/01_-_Convolutional_layers_1D.py

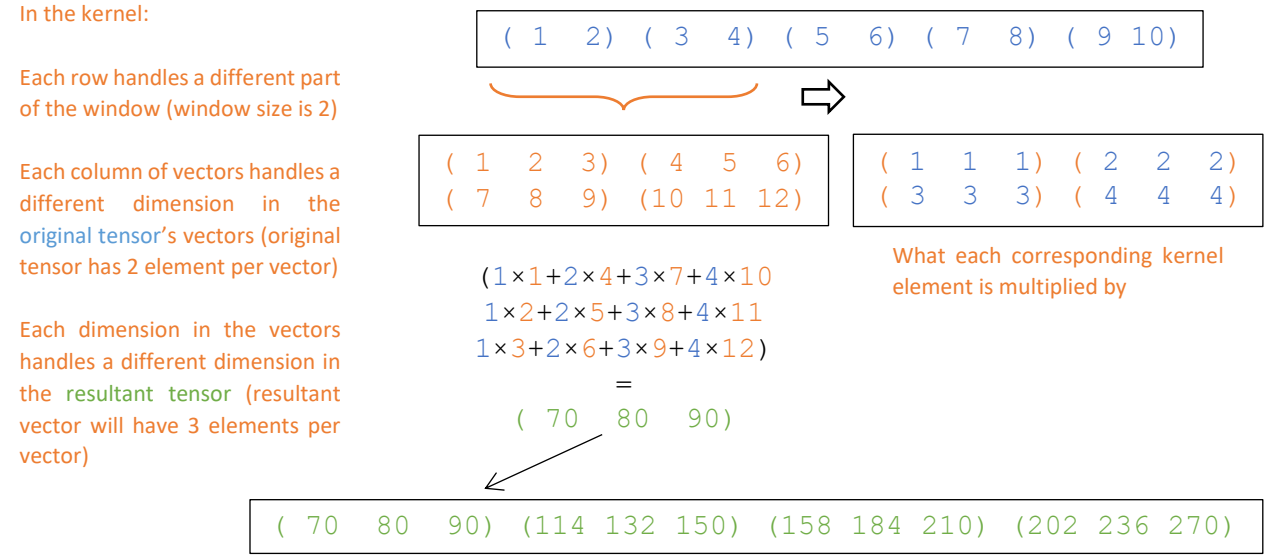
A convolution is just a mathematical operation that involves sliding a tensor over a larger tensor of the same dimensionality. The smaller tensor is called a kernel. At every position that the kernel is slid, you multiply the overlapping elements of the two tensors and add the products. The result is placed in the corresponding position in a resultant tensor. Here is an example using vectors, also known as a 1D convolution, with the kernel in orange, the tensor being convolved in blue, and the resultant tensor in green:



You can also control the stride of the convolution, that is, the amount by which the kernel moves after each step, and the padding in order to slide outside the edges of the larger tensor ('VALID': no padding, 'SAME': pad the edges of the larger tensor with zeros).

In neural networks we assume that a sequence consists of vectors rather than scalars, which means that a 1D convolution is actually applied to a matrix rather than a vector. The resultant tensor would also be a matrix. In practice the sequence could be a sentence with each vector in the sequence being a word vector.

Convolutions become more complex when we consider that, although the resultant tensor will have the same number of dimensions as the original tensor, the size of the dimensions can be changed after a convolution. In fact a kernel should have three dimensions: the first dimension is for the window width (in the above diagram it was three), the second dimension is for the vector size in the original tensor (in the above diagram it has a vector size of one), and the third dimension is for the vector size in the resultant tensor (in the above diagram it has a vector size of one). Below is an illustration:



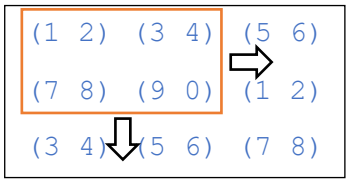
2. Convolutional layers 2D

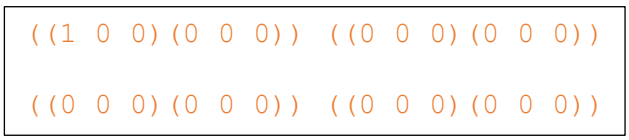
https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/02_-_Convolutional_layers_2D.py

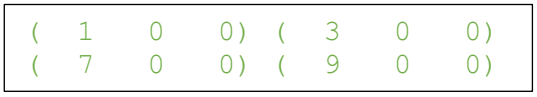
We can also apply convolutions on grids rather than just linear sequences. This would be called a 2D convolution. In 2D convolutions the kernel would slide along the two dimensions rather than slide along one dimension as was the case above.

Again, in neural networks we assume that a grid consists of vectors so we're really applying the convolution on a 3D tensor rather than a matrix. The resultant tensor would also be a 3D tensor. In practice the grid could be an image with each vector in the grid being the red-green-blue channels of each pixel.

Note that the kernel will now have four dimensions instead of three as there is now a width and a height instead of just a width. The first dimension is for the window height, the second dimension is for the window width, the third dimension is for the vector size in the original tensor, and the fourth dimension is for the vector size in the resultant tensor. Below is an illustration:







In the kernel:

Each quadrant handles a different part of the window (window size is 2x2)

Within each quadrant, each column of vectors handles a different dimension in the original tensor's vectors (original tensor has 2 element per vector)

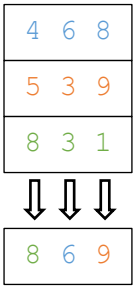
Each dimension in the vectors handles a different dimension in the resultant tensor's vectors (resultant vector will have 3 elements per vector)

3. CNN 1D pooled

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/03_-_CNN_1D_pooled.py

One way to use a convolutional operation in a neural network is to treat the sliding window's numbers as weights that need to be learned. The input is first convolved, then added a bias, and finally passed through an activation function. When a neural network makes use of a convolution layer, the neural network is often called a convolutional neural network (CNN).

Following the convolution layer, the next step in a CNN is to turn the convolution layer's result (activations) into an output of the neural network. The layer's result must be converted into a single vector in order to then pass it to a sigmoid or softmax layer. One way to do this is to squash all the individual vectors being produced for each sliding window position into one vector by max pooling, meaning that you take the maximum value at each corresponding position in all the vectors like this:



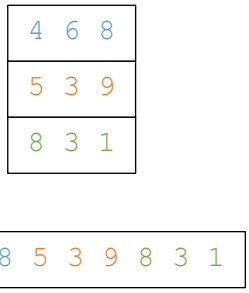
If each value in the vector is taken to be a binary flag which says whether an important feature in the input is present or not (such as whether a trigram that is typically associated with positive or negative sentiment is present in the sentence) then taking the maximum of all the corresponding flags is equivalent to checking if at least one of the positions visited by the sliding window contains the important feature.

In this code there is a 1D CNN performing sentiment analysis, that is, classifying text on whether it expressives positive sentiment or not. It will decide this using a sliding window of two words.

4. CNN 1D fully connected

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/04_-_CNN_1D_fully_connected.py

Another way to convert the convolution layer's result into a single vector is to just concatenate all the vectors into a single giant vector, a process called flattening, like this:



The disadvantage of this method is that it forces the sentences to always have the same number of words since the final flat vector must have a fixed size so that the output layer can multiply it by a weight matrix. On the other hand it does away with the concept of independent flags that indicate whether there is some important feature or not and instead allows for all the convolved vectors to interact together and take into account the position of where the feature was located.

5. CNN 2D pooled

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/05_-_CNN_2D_pooled.py

Although convolutions are used for reading sentences, their typical use is in reading images. Here we process 3D tensors (images with vector pixels) instead of matrices (sentences with vector words) using max pooling again.

In the code we will be classifying 3 by 3 pixel images containing shapes and the task is to classify whether the shape is a single line. We shall use a 2 by 2 sliding window to do this. Notice how this neural network cannot learn that the empty image has no lines because it can only use independent 2 by 2 square windows to determine whether there is a line or not and just because there is nothing in one square does not mean that there is nothing in the whole image.

6. CNN 2D fully connected

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/05_-_CNN_2D_fully_connected.py

Max pooling is not typically used to turn image convolutions into fixed vectors. Instead they are usually flattened. This is less of a problem for images than sentences because images can be resized to always have the same width and height.

7. CNN 2D fully connected 2 layer

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/07_-_CNN_2D_fully_connected_2_layer.py

A convolution layer can process the result of another convolution layer. Practical object recognition neural nets would typically have a number of convolution layers in sequence. This is easy to do because the result of a convolutional layer has the same number of dimensions as its input and so can be processed as if it's another image or sentence.

8. CNN 2D down sampled

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/06_-_Convolutional_neural_networks/08_-_CNN_2D_down_sampled.py

Since images typically have a lot pixels, the neural network will tend be big and slow. To reduce this problem, each convolution layer can resize its result by dividing the resultant tensor into windows (with no overlap) and then taking the maximum value of each individual window and using it to replace the window. This is called down sampling (resizing to be smaller) and results in a convolution result that is a fraction of the original size. Generally, practical neural networks would consist of three stages per layer: a convolution, a ReLU, and a down sample.