Linear Operators in a Generative Adversarial Network

Jasper Hill

February 24, 2020

1 Introduction

The quest to emulate human-like intelligence in computers has been aided significantly by the employment of convolutional neural networks (CNNs). In accordance with its name, a CNN features one or more layers that act to convolve each input channel of a tensor with a kernel to generate some number of output channels with altered shapes. Though the success of CNNs is unambiguous, their mathematical foundation seems a bit unsound. The purpose of this experiment is to develop an alternative set of tools and to investigate their efficacy within the context of a generative adversarial network.

2 Convolutional Layers

A specific element of the convoluted tensor, $\mathbf{A}'$, can be approximately expressed in terms of the input tensor, $\mathbf{A}$, and convolution operator, $\hat{C}$:

$$\mathbf{A}'^{(n_o)}_{ij} = (\hat{C}\mathbf{A})^{(n_o)}_{ij} = \Big(\sum_{n_i}^{N_i} \hat{C}^{(n_o)} * A^{(n_i)}\Big)_{ij} = \sum_{n_i}^{N_i} \sum_{kl}^{KL} c^{(n_o)}_{kl} A^{(n_i)}_{i\cdot\Delta+k,j\cdot\Delta+l}. \tag{1}$$

Here, n_o and n_i are the output and input channel indices respectively, $*$ is the Hadamard operator in $K \times L$ space, and the stride length is Δ. It is clear from the rightmost expression that each transformed tensor element encodes a limited amount of information from the original tensor. In order to correlate tensor elements respectively separated by heights or widths greater than $\Delta + K$ or $\Delta + L$, multiple convolution layers must be successively applied. For this reason, attaining highly-sophisticated functionality, such as image segmentation or style transfer, requires increasingly many layers to couple each of the original tensor elements with one another.

3 Linear Operators

Unlike the convolutional layer, which employs the Hadamard product between its kernels and the input matrices, the linear operators utilized in this experiment facilitate a transformation on the full space of the input tensors:

$$
\begin{aligned}
\mathbf{A}'^{(n_o)}_{ij} &= (\hat{O}\mathbf{A}\hat{P})^{(n_o)}_{ij} \\
&= \Big(\sum_{n_i}^{N_i} \hat{O}^{(n_o,n_i)} A^{(n_i)} \hat{P}^{(n_o,n_i)} \Big)_{ij} \\
&= \sum_{n_i}^{N_i} \sum_{kl}^{MN} O^{(n_o,n_i)}_{ik} A^{(n_i)}_{kl} P^{(n_o,n_i)}_{lj}
\end{aligned}
\tag{2}
$$

for $A \in \Re^{M\times N}$ with $O \in \Re^{\times M' \times M}$ and $P \in \Re^{N\times N'}$. The output, $\mathbf{A}'$, then, will contain N_o matrices of dimension $M' \times N'$. Importantly, each element of the transformed tensor now encodes all information from the input. In theory, all necessary spatial correlations can be learned with only one layer.

4 Tensor Flattening

When a network is used to generate scalar information from two-dimensional images, the common strategy is to employ a series of convolutional layers, and, when a sufficient number of output channels has been produced, the resulting tensors are unpacked so that they are of rank 1 (i.e., vectors). It is assumed then, that any important spatial correlation among these vector elements can be learned by the input weights of the remaining dense layers. A more mathematically-rigorous way to ensure this is to integrate the image over vectors $\vec{u}$ and $\vec{v}$:

$$\vec{y} = \vec{u}^{\dagger} A \vec{v} \tag{3}$$

This operation converts the rank-3 tensor, $\mathbf{A}$, into the vector, $\vec{y}$, whose elements are given by

$$y^{(n_i)} = \sum_{ij}^{MN} u^{(n_i)}_i A^{(n_i)}_{ij} v^{(n_i)}_j . \tag{4}$$

In this way, the 2-dimensional structure of the input matrices is integrated into each element of the flattened tensor. Importantly, this operation can be implemented with the same machinery described above with $O \in \Re^{1\times M}$ and $P \in \Re^{N\times 1}$