1 Median Filtering

Recall that a convolutional kernel in image contexts involves calculating the sum of the elementwise products of the kernel with a window of pixels in the image, then sliding the kernel across the image, repeating the process for every pixel in the image.

(a) Consider the following 3x3 kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

This is often referred to as a "box-blur" kernel. Why does it have a blurring effect?

The box-blur kernel effectively averages each pixel with its neighbors. This means that the resulting image will be blurred, because sharp edges will become somewhat gradiented.

(b) Now consider a related technique, known as *median filtering*. Here, for a given window of pixels, the center pixel is replaced with the median of the pixels in the window. This process is then repeated across the entire image, just as is done with convolution.

For your curiosity, depicted below are the results of applying 3x3 filters to an image (original left, box-blur center, median right):

Can you think of a scenario where median filtering would be preferable to box-blur filtering? What about the other way around?

Median filtering can be very useful for denoising, especially where the noise is extremely pronounced, because it eliminates outliers, whereas box-blur filtering will be largely affected by outliers. It also retains edges better than box-blur. Box-blur filtering is useful for universally blurring images, and is also easier to compute.

(c) For ConvNets, are median filters similar to convolutional layers? Are they still useful? Why or why not?

Median filters are not similar to convolutional layers, because they do not have learnable parameters. In this sense, they are more similar to pooling layers. However, they can certainly still be useful, e.g. in preprocessing steps, because of their denoising capabilities.

(d) Consider the infinite zero matrix

$$\mathbf{0} = \begin{bmatrix} \ddots & \vdots & \vdots & \iddots \\ \cdots & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \cdots \\ \iddots & \vdots & \vdots & \ddots \end{bmatrix},$$

which, for this problem, represents a bitmap of constant color. Now suppose each pixel in the image is independently at random flipped to 1 with probability $p < 0.2$, representing random noise. What is the expected result of applying a 3×3 box-blur filter to this newly noised image? What about a 3×3 median filter? Assume the stride length is at least 3 (so there is no overlap).

Do your answers to (b) and (c) now change?

The expectation of the noise is p times the all-ones matrix (denoted $\mathbf{J}$), and a box-blur filter simply averages all the pixels in its window, so the expected result of applying a box-blur filter is

$$\frac{1}{9}(\mathbf{0} + p \cdot \mathbf{J}) = \frac{p}{9} \cdot \mathbf{J}.$$

With a median filter, for a pixel to change, 1 must be the new median of the window after the noise is applied, which means that at least 5 pixels must be flipped to 1 in any given 3×3 window. The probability q of this occurring is

$$\begin{aligned} q &:= \sum_{k=5}^{9} \binom{9}{k} p^k (1-p)^{9-k} \\ &= p^9 + 9(1-p)p^8 + 36(1-p)^2 p^7 + 84(1-p)^3 p^6 + 126(1-p)^4 p^5. \end{aligned}$$

The expected result of applying the median filter is then

$$(\mathbf{0} + q \cdot \mathbf{J}) = q \cdot \mathbf{J}.$$

For $p < 0.2$, $q < \frac{p}{9}$ (and the difference grows much faster as $p \to 0$), so if the goal is to restore the original image, median filtering performs better at this denoising task than box-blur filtering.

(e) Let $K \times K$ denote filter size, P padding size, S stride length, and suppose we are operating on an image of size $H \times W$. What is the runtime complexity of convolution? How about median filtering? Assume no results are cached across different pixels.

Hint: You can find the median of n numbers in $O(n)$ time.

The image after padding is effectively $(H + 2P) \times (W + 2P)$. Stride reduces computation by approximately $1/S^2$, because each $S \times S$ window has only one elementwise product operation. For each pixel considered, there are K^2 multiplications. Thus, the complexity of convolution is $O(\frac{K^2}{S^2}(H+P)(W+P))$. However, padding is always less than the size of the image, so the complexity is $O(\frac{K^2 HW}{S^2})$. The complexity of median filtering is the same, because both the mean and the median take linear time to compute.

2 Weighted Cross-Entropy

Recall that cross-entropy loss is given by

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}),$$

where y is the ground truth label and $\hat{y}$ is the predicted probability that the label is 1.

(a) Suppose p is the ground-truth probability of a positive sample. Find the value of $\hat{y}$ that minimizes the expected loss

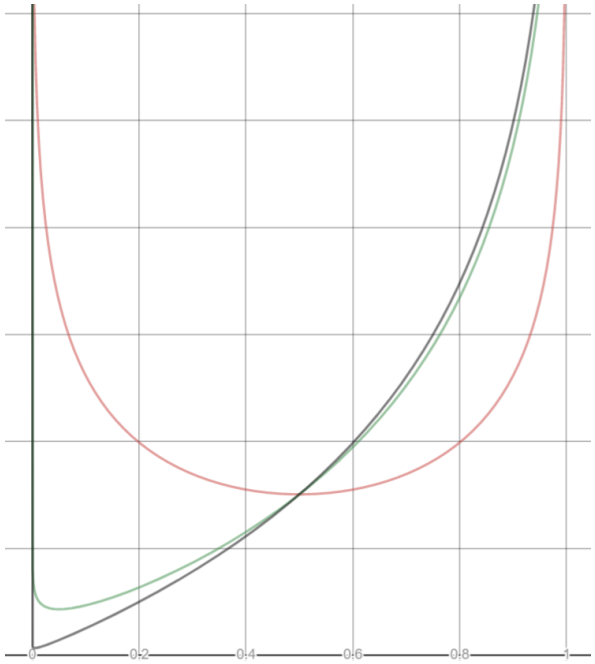
$$\mathbf{E}_{y \sim \text{Bernoulli}(p)} \left[\mathcal{L}(y, \hat{y}) \right]$$

by taking the derivative with respect to $\hat{y}$ and setting it to zero.

The derivative is given by

$$\begin{aligned}
\frac{\partial}{\partial \hat{y}} \mathbf{E}_{y \sim \text{Bernoulli}(p)} \left[\mathcal{L}(y, \hat{y}) \right] &= 0 \\
\frac{\partial}{\partial \hat{y}} (-p \log \hat{y} - (1 - p) \log(1 - \hat{y})) &= 0 \\
-\frac{p}{\hat{y}} + \frac{1 - p}{1 - \hat{y}} &= 0 \\
\frac{1 - p}{1 - \hat{y}} &= \frac{p}{\hat{y}} \\
\hat{y}(1 - p) &= p(1 - \hat{y}) \\
\hat{y} - p\hat{y} &= p - p\hat{y} \\
\hat{y} &= p.
\end{aligned}$$

(b) Now, consider how the model is affected by p. Below is the graph of $\mathbf{E}[\mathcal{L}]$ as a function of $\hat{y}$ for $p = 0.5$, $p = 0.05$, and $p = 0.005$. From this, comment on what problems may arise when using binary cross-entropy loss (you do not need to show these rigorously).



Answers may vary. Our answer is as follows: For extremely unbalanced distributions (where p is very close to 0 or 1), the optimal value of $\hat{y} = p$ will be difficult to attain because it is very easy to overshoot. This means that in order to minimize loss, the learning rate will need to be extremely small, slowing training significantly. Additionally, the model may simply draw an affinity for the majority class, always outputting $\hat{y} = 1$ or $\hat{y} = 0$, failing to generalize to the minority class.

(c) Suppose we want to modify the loss function to be more robust to unbalanced distributions. One way to do this is to introduce a hyperparameter α, which we can use to penalize incorrect predictions more heavily for the minority class (or equivalently, less heavily for the majority class). This is called *weighted cross-entropy* and is given by

$$\mathcal{L}_\alpha(y, \hat{y}) := -\alpha y \log \hat{y} - (1 - y) \log(1 - \hat{y}).$$

(i) Find the value of $\hat{y}$ that minimizes the expectation of this loss. Remember that the ground-truth probability of a positive sample is p, and $y \sim \text{Bernoulli}(p)$.

Similarly to before, we take the derivative and set it to zero.

$$\begin{aligned}
\frac{\partial}{\partial \hat{y}} \mathcal{L}_\alpha(y, \hat{y}) &= 0 \\
\frac{\partial}{\partial \hat{y}} (-\alpha p \log \hat{y} - (1 - p) \log(1 - \hat{y})) &= 0 \\
-\frac{\alpha p}{\hat{y}} + \frac{1 - p}{1 - \hat{y}} &= 0 \\
\frac{1 - p}{1 - \hat{y}} &= \frac{\alpha p}{\hat{y}} \\
\hat{y}(1 - p) &= \alpha p(1 - \hat{y}) \\
\hat{y} - p\hat{y} &= \alpha p - \alpha p \hat{y} \\
(1 - p + \alpha p)\hat{y} &= \alpha p \\
\hat{y} &= \frac{\alpha p}{1 - p + \alpha p}
\end{aligned}$$

(ii) Use the value of $\hat{y}$ from part (i) to find α in terms of p such that, when averaged over sufficiently many samples, both the minority and majority classes contribute to the total loss equally.

Note: This derivation is quite difficult, but the final answer should be relatively simple. Feel free to use a symbolic or graphing calculator.

We want the loss to be equal for both the positive and negative classes, so we set

$$\begin{aligned}
\alpha p \log \hat{y} &= (1 - p) \log(1 - \hat{y}) \\
\alpha p \log \left(\frac{\alpha p}{1 - p + \alpha p} \right) &= (1 - p) \log \left(1 - \frac{\alpha p}{1 - p + \alpha p} \right) \\
\alpha p \log \left(\frac{\alpha p}{1 - p + \alpha p} \right) &= (1 - p) \log \left(1 - \frac{(1 - p + \alpha p) - (1 - p)}{1 - p + \alpha p} \right) \\
\alpha p \log \left(\frac{\alpha p}{1 - p + \alpha p} \right) &= (1 - p) \log \left(\frac{1 - p}{1 - p + \alpha p} \right) \\
\alpha p \left(\log(\alpha p) - \log(1 - p + \alpha p) \right) &= (1 - p) \left(\log(1 - p) - \log(1 - p + \alpha p) \right) \\
\alpha p \log(\alpha p) - \alpha p \log(1 - p + \alpha p) &= (1 - p) \log(1 - p) - (1 - p) \log(1 - p + \alpha p) \\
(1 - p - \alpha p) \log(1 - p + \alpha p) &= (1 - p) \log(1 - p) - \alpha p \log(\alpha p)
\end{aligned}$$

Let $q = 1 - p$, then $p = 1 - q$, so

$$(q - \alpha(1-q))\log(q + \alpha(1-q)) = q\log q - \alpha(1-q)\log(\alpha(1-q))$$

Then let $r = \alpha(1-q)$, so

$$(q-r)\log(q+r) = q\log q - r\log r$$

Note that the equation is invariant to swapping q and r. This can be seen as follows:

$$\begin{aligned}(q-r)\log(q+r) &= q\log q - r\log r \\ (r-q)\log(r+q) &= -(q-r)\log(q+r) = -(q\log q - r\log r) = r\log r - q\log q\end{aligned}$$

This suggests a symmetry over $q = r$, and indeed, it can be verified that $q = r$ satisfies the equation. This yields $q = r$ and we can solve for α:

$$\begin{aligned}q &= r \\ q &= \alpha(1-q) \\ \alpha &= \frac{q}{1-q} \\ \alpha &= \frac{1-p}{p}\end{aligned}$$

3 Basic Pitch

In 2022, Spotify released a free software known as Basic Pitch that can translate raw audio files to digitized sheet music (known as MIDI). You can find the paper at `https://arxiv.org/pdf/2106.02769.pdf`.

(a) The paper describes median filtering as a common technique for denoising audio, using a 1D median filter (audio files are essentially one-dimensional time series of frequencies). Suppose we only had access to square median filters. How can we use these square median filters to denoise audio samples?

Hint: How do we generally visualize functions of a single variable? Can you apply that here?

Using a 1×1 median filter here doesn't work because it doesn't modify anything, so no denoising will occur. We can, however, plot frequency against time (known as a *spectrogram*), and then apply a median filter. We can then convert the denoised spectrogram back to audio by aggregating the resulting pixel values for each time step (by, for example, averaging) to collapse it into a 1D vector.

One problem with the above is that the median filter may see the true signal as noise, but this can be mitigated by using coarser bins to generate the spectrogram (so multiple pixels on the frequency-axis are positive for the true signal).

(b) Basic Pitch uses various sophisticated losses to determine the quality of the generated MIDI files, but they are all binary. For example, consider the question "Is the note at a given frequency being played?" How does this relate to what you learned in problem (2)?

For this specific question (and other ones used in the paper), it is much more likely to generate an negative sample than a positive one (there are simply many more negative answers, because most sheet music have relatively few frequencies being played compared to those not being played). For example, a piano has 88 keys, but it is essentially impossible to play more than 44 keys simultaneously. As a result, there will be many more negative results than positive ones, and the model may skew towards always predicting a negative value. To account for this, the authors use the weighted cross-entropy loss described in problem (2) with the goal of having the model reward positive predictions much more than negative ones.

(c) Complete the associated HW Jupyter notebook. Comment on what you've learned.

Answers will vary.