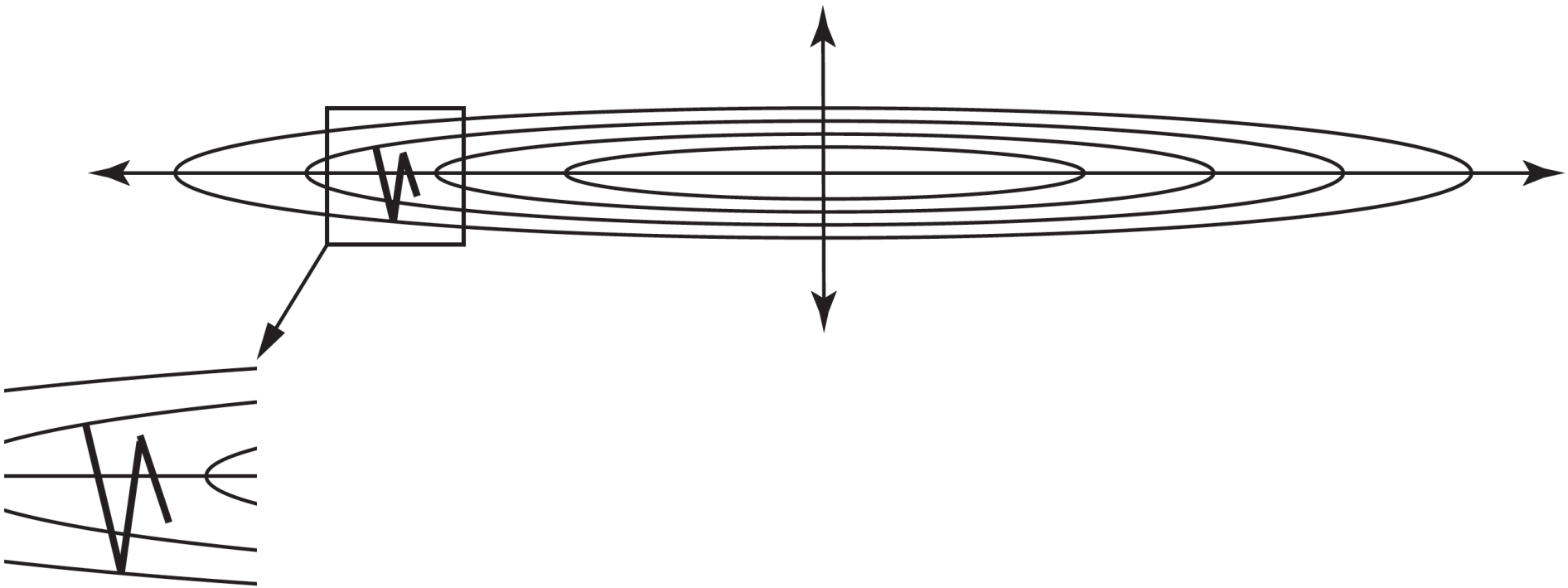
Building a working autoencoder – III Optimization tricks

D.A. Forsyth

University of Illinois at Urbana Champaign

Gradients aren’t ideal…



Gradients aren't ideal...

Here is an example in algebra. Consider $f(x, y) = (1/2)(\epsilon x^2 + y^2)$, where ϵ is a small positive number. The gradient at (x, y) is $(\epsilon x, y)$. For simplicity, use a fixed learning rate η, so

$$\begin{bmatrix} x^{(r)} \\ y^{(r)} \end{bmatrix} = \begin{bmatrix} (1-\epsilon\eta)x^{(r-1)} \\ (1-\eta)y^{(r-1)} \end{bmatrix}.$$

Start at, say, $(x^{(0)}, y^{(0)})$ and repeatedly go downhill along the gradient; you will travel very slowly to your destination. You can show that

$$\begin{bmatrix} x^{(r)} \\ y^{(r)} \end{bmatrix} = \begin{bmatrix} (1-\epsilon\eta)^r x^{(0)} \\ (1-\eta)^r y^{(0)} \end{bmatrix}.$$

The problem is that the gradient in y is quite large (so y must change quickly) and the gradient in x is small (so x changes slowly). In turn, for steps in y to converge requires $|1-\eta| < 1$; but for steps in x to converge requires only the much weaker constraint $|1-\epsilon\eta| < 1$. Choose the largest η you dare for the y constraint. The y value will very quickly have small magnitude, though its sign will change with each step. But the x steps will move closer to the right spot only extremely slowly.

But you can't do Newton's method

- Too many variables; too expensive

- Insight:

One useful insight into the problem is that fast changes in the gradient vector are worrying. For example, consider $f(x) = (1/2)(x^2 + y^2)$. Imagine you start far away from the origin. The gradient won't change much along reasonably sized steps. But now imagine yourself on one side of a valley like the function $f(x) = (1/2)(x^2 + \epsilon y^2)$ (Figure 18.7); as you move along the gradient, the gradient in the x direction gets smaller very quickly, then points back in the direction you came from. You are not justified in taking a large step in this direction, because if you do you will end up at a point with a very different gradient. Similarly, the gradient in the y direction is small, and stays small for quite large changes in y value. You would like to take a small step in the x direction and a large step in the y direction.

Momentum

- Smooth the gradient using moving average

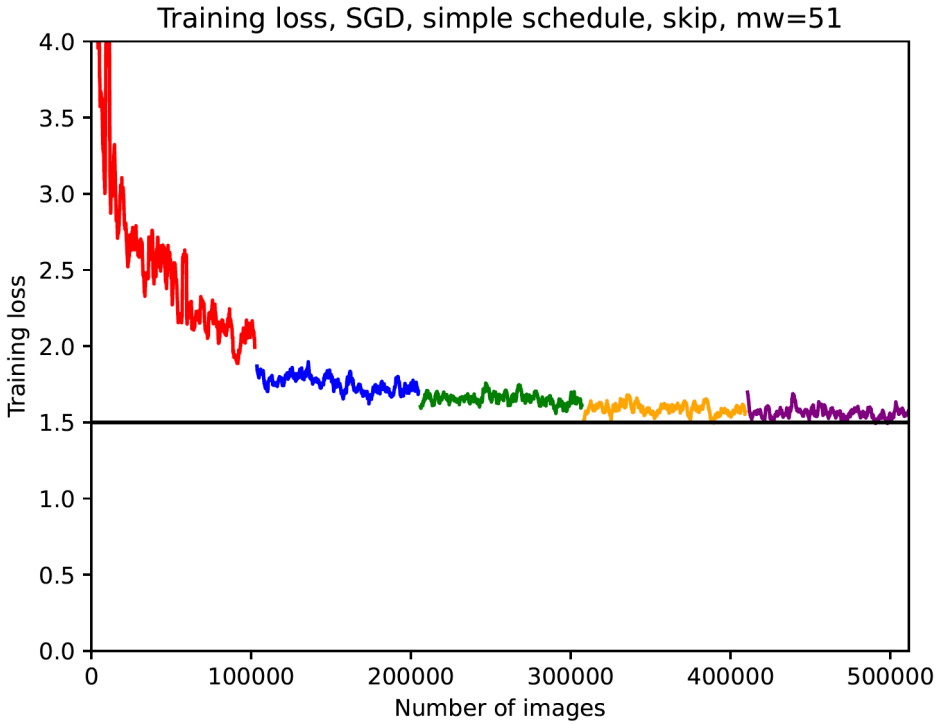
Procedure: 18.1 *Momentum*

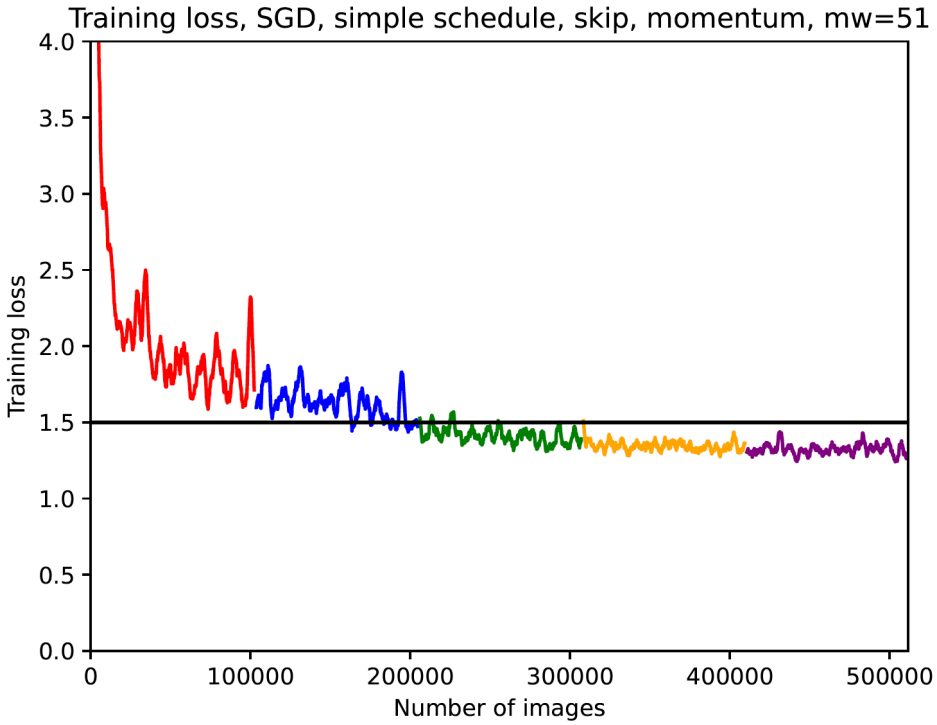
Construct a vector $\mathbf{v}$, the same size as the gradient, and initialize this to zero. Choose a positive number $\mu < 1$ (the momentum). Then iterate

$$\begin{aligned} \mathbf{v}^{(r+1)} &= \mu\mathbf{v}^{(r)} + \nabla_\theta E \\ \theta^{(r+1)} &= \theta^{(r)} - \eta\mathbf{v}^{(r+1)} \end{aligned}$$

Larger μ means the final update sees a moving average of the gradient over a longer window.

Momentum helps





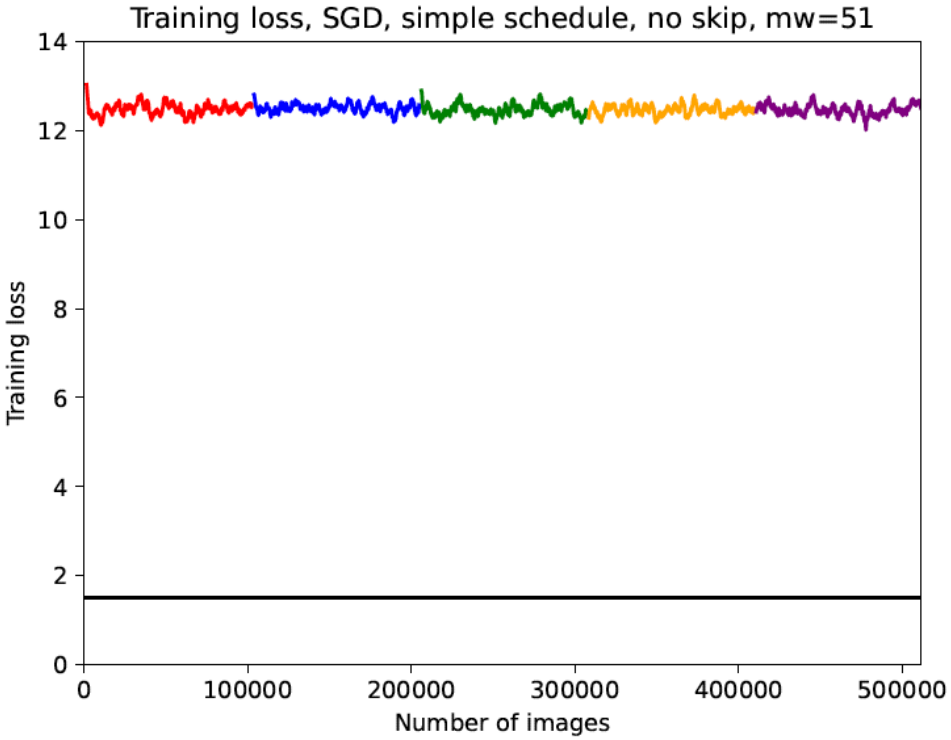
Adam – momentum variant

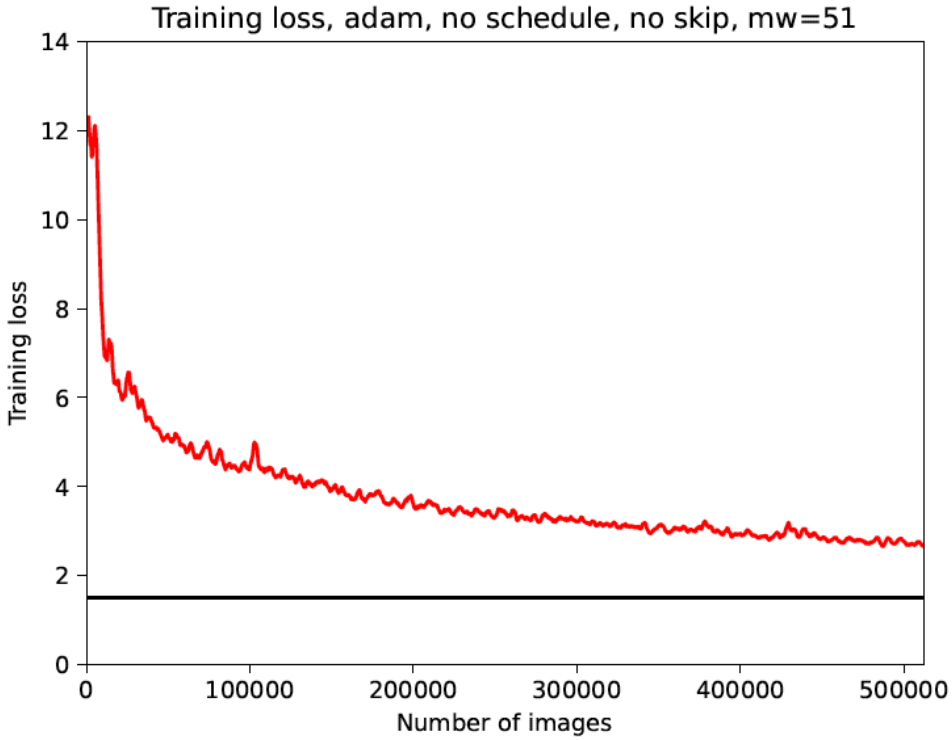
Procedure: 18.2 *Adam*

Write $g_i^{(r)}$ for the i'th component of the gradient $\nabla_\theta E$ computed at the r'th iteration. Choose three numbers β_1, β_2 and ϵ (typical values are 0.9, 0.999 and 1e-8, respectively), and some stepsize or learning rate η, then iterate

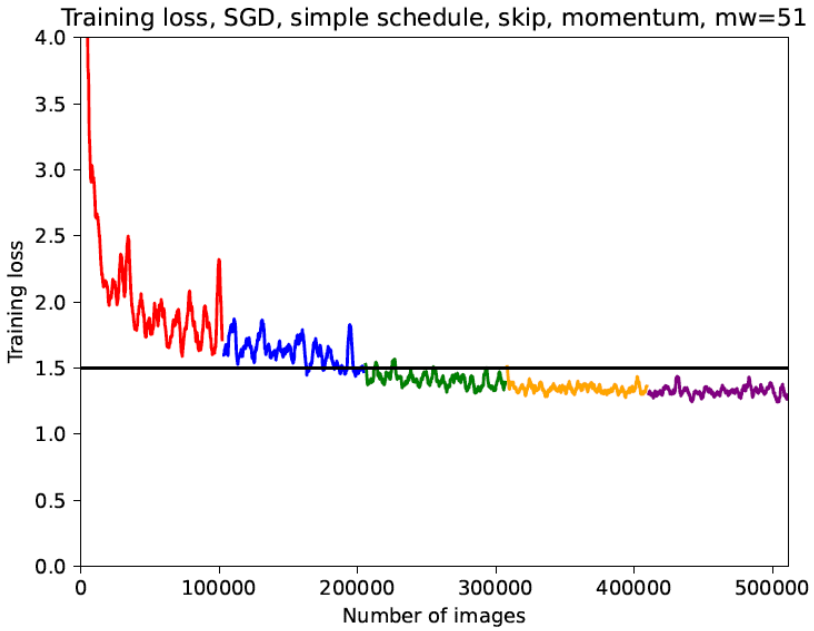
$$
\begin{aligned}
\mathbf{v}^{(r+1)} &= \beta_1 * \mathbf{v}^{(r)} + (1-\beta_1) * \nabla_\theta E \\
c_i^{(r+1)} &= \beta_2 * c_i^{(r)} + (1-\beta_2) * (g_i^r)^2 \\
\hat{\mathbf{v}} &= \frac{\mathbf{v}^{(r+1)}}{1-\beta_1^t} \\
\hat{c}_i &= \frac{\hat{c}_i^{(r+1)}}{1-\beta_2^t} \\
\theta_i^{(r+1)} &= \theta_i^{(r)} - \eta \frac{\hat{v}_i}{\sqrt{\hat{c}_i} + \epsilon}
\end{aligned}
$$

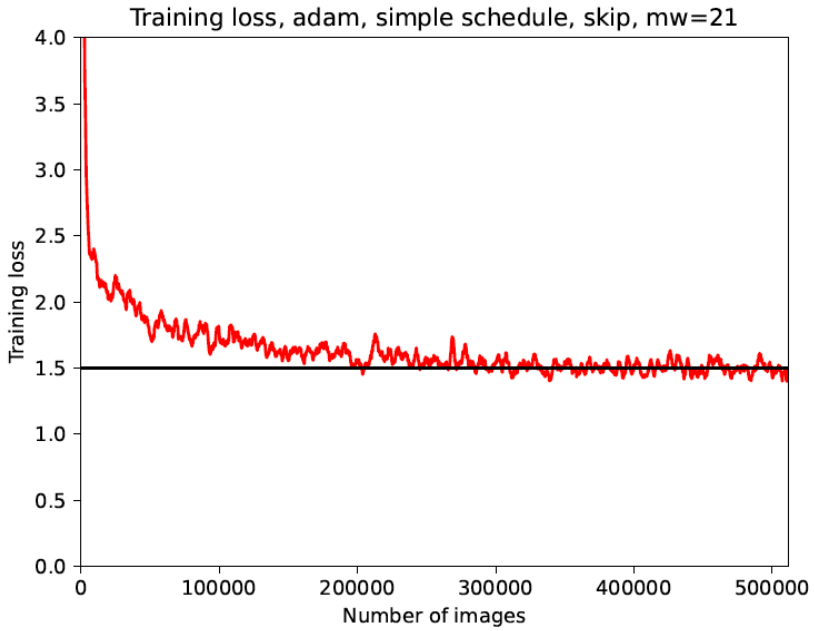
Adam can help a lot



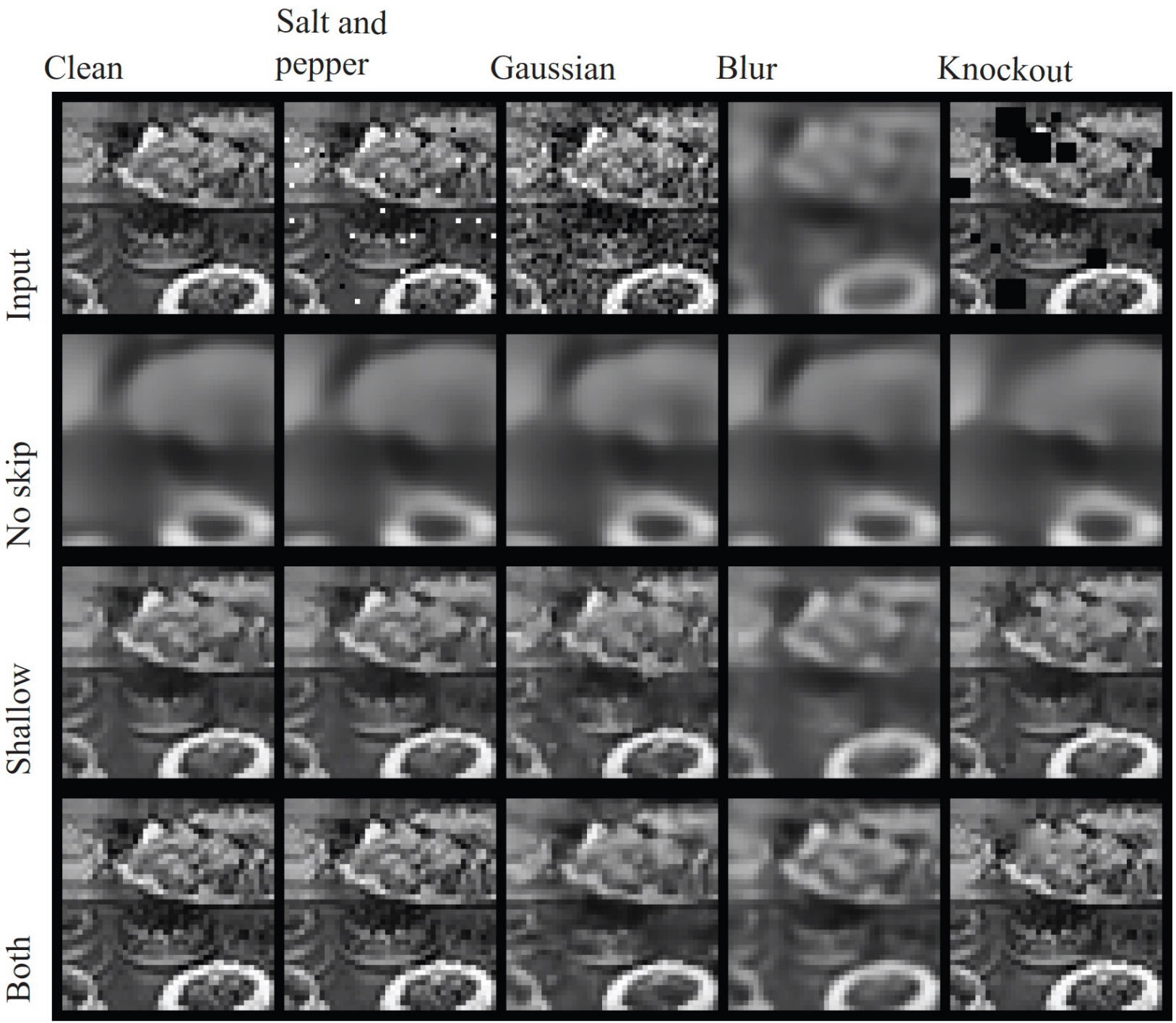


but isn’t perfect...





Clean
Salt and pepper
Gaussian
Blur
Knockout
PSNR:
Mean (std); this
21 (2.8); 17
26 (3); 24
30 (4); 29
Input
No skip
Shallow
Both


Clean
Salt and pepper
Gaussian
Blur
Knockout
Input
No skip
Shallow
Both

Things to think about

18.1. A sigmoid

$$\text{sigmoid}(x) = \frac{e^x}{1+e^x}$$

maps $[-\infty, \infty]$ to $[0, 1]$. What is the derivative of a sigmoid? When $\text{sigmoid}(x)$ is close to 1 or to 0, what can you say about the derivative? Why might this be a nuisance?

18.2. Write

$$f(x) = \text{ReLU}(x) - \text{ReLU}(x-1).$$

Show $f(x) = 0$ for $x \leq 0$ and $f(x) = 1$ for $x \geq 1$. What is the gradient of $f(x)$ in these cases?

18.3. Section 18.3.2 has: "Even though real layers involve non-linearities (at least, a ReLU here and there), this linear model is informative. This is because you could linearize the network about the current operating point, and the gradient of that linearized network is the true gradient at that operating point." Explain – why can you get away with linearizing about an operating point when a ReLU isn't differentiable?

18.4. Section ?? has: " **Warning:** A reliable source of errors in using an API is not to "tell" the environment that you want a network with batch normalization layers in it to switch from training to evaluation mode – be careful about this." Explain.

18.5. Sophisticated learning rate schedules mostly decrease the learning rate, but sometimes *increase* it. Why might occasionally increasing the learning rate be a good idea?