Scientific Computing in Python / 2027 Lecture 5

Neural-network gradients and spatial filters

Lecture notes

An image is an array of measured intensities. We first treat that array as a vector and derive the forward and backward passes of a small neural network. We then retain its spatial structure and apply local filters. The synthetic stripe problem makes both computations explicit, while keeping the distinction between fixed filters and learned parameters visible.

1 Images as arrays

Slides for this section

An 8×8 single-channel image contains 64 scalar intensities. Flattening produces a vector with a fixed ordering. The ordering does not discard numerical entries, but a dense layer does not explicitly encode which pixels are adjacent. A local filter uses spatial indices to reuse the same coefficients across positions.

$$X \in \mathbb{R}^{n\times 8\times 8\times 1}, \qquad X_{\text{flat}} \in \mathbb{R}^{n\times 64}.$$

Flattening preserves the values and fixes a coordinate convention.

An affine hidden layer

$$U = X_{\text{flat}} W_1 + b_1, \quad H = \tanh U,$$

$$z = H w_2 + b_2, \qquad p = \sigma(z).$$

For h hidden units, W_1 has shape $64 \times h$.

2 The binary loss

Slides for this section

For binary labels the negative Bernoulli log likelihood can be written in terms of logits. This avoids directly taking logarithms of scores that have rounded to zero or one. Averaging over the training batch makes the data term's scale independent of batch length.

$$L = \frac{1}{n} \sum_i [\log(1 + e^{z_i}) - y_i z_i].$$

```
loss = np.mean(np.logaddexp(0, z) - y*z)
```

Derivative with respect to the logits

$$\frac{\partial L}{\partial z_i} = \frac{p_i - y_i}{n}.$$

The derivative is bounded even when a logit has large magnitude. Stable evaluation remains necessary.

3 Backpropagation

Slides for this section

The chain rule propagates the loss derivative from logits to hidden states and then to the first affine map. Tanh has derivative $1 - H^2$, applied entrywise. Matrix shapes expose which indices are being summed. Penalty terms, if included in the objective, contribute additional parameter derivatives.

Let $D = (p - y)/n$. Then

$$\nabla_{w_2} L = H^\mathsf{T} D, \qquad D_1 = (D w_2^\mathsf{T}) \odot (1 - H^2).$$

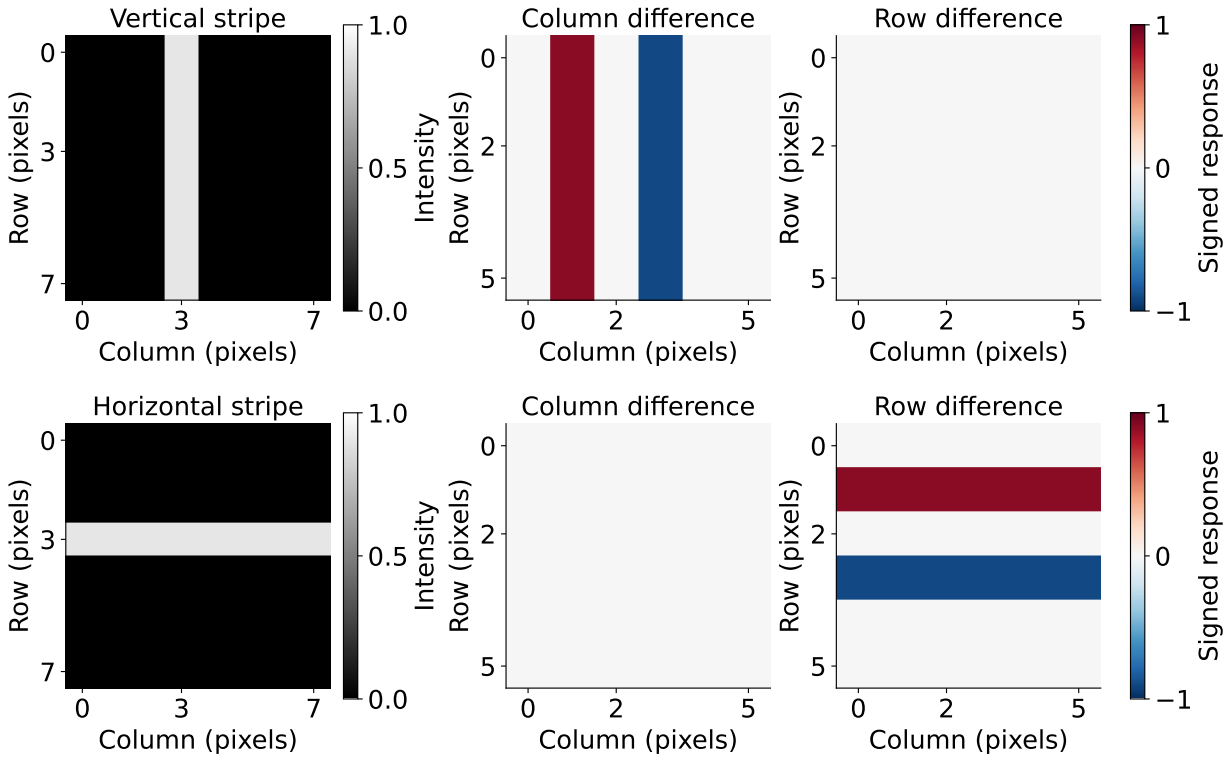
$$\nabla_{W_1} L = X_{\text{flat}}^\mathsf{T} D_1.$$

A finite-difference gradient check

For a parameter coordinate θ_j,

$$\partial_j L \approx \frac{L(\theta + \epsilon e_j) - L(\theta - \epsilon e_j)}{2\epsilon}.$$

Use a moderate ϵ: too large gives truncation error, too small gives cancellation error.



Separate idealized 8×8 display images place each stripe in the interior so both edge signs remain visible in the 6×6 responses. These fixed 3×3 filters use cross-correlation with no padding; red and blue show opposite signs. The images are not benchmark samples, and the filters are not learned.

4 Local cross-correlation

Slides for this section

A 3×3 filter applied without padding to an 8×8 image produces a 6×6 response. The implemented operation is cross-correlation because it does not reverse the filter indices. Deep-learning libraries often call this operation convolution. Its boundary convention must be stated because padding changes both shape and edge behavior.

$$R_{nijf} = \sum_{u=0}^{2} \sum_{v=0}^{2} X_{n,i+u,j+v,0} K_{uv0f}.$$

The output shape is $(n, 6, 6, F)$ for F filters.

Output size

For input width W, filter width k, padding p, and stride s,

$$W_{\text{out}} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1.$$

Here $W = 8$, $k = 3$, $p = 0$, $s = 1$, giving 6.

5 Filter response and orientation

Slides for this section

A filter with coefficients summing to zero gives zero response on a constant patch. A signed edge filter responds differently to increasing and decreasing intensity. Reversing the edge reverses the sign. The filters in this demonstration are fixed by construction, while the dense network parameters are fitted from labeled images.

$$K = \frac{1}{3}\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, \qquad \sum_{u,v} K_{uv} = 0.$$

A constant patch has response $c \sum K_{uv} = 0$.

Parameter sharing

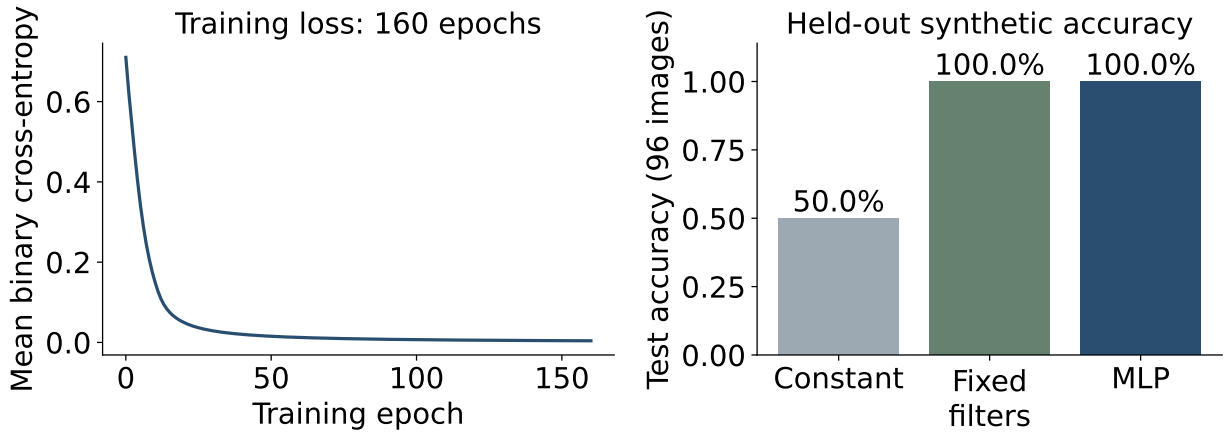
The same 3×3 coefficients act at every valid image position.

This reuses nine parameters per input/output channel pair. A dense map would assign separate weights to each input coordinate for each output.

A different pointwise activation, used in the exercises, is the rectified linear unit:

$$\mathrm{ReLU}(a) = \max(0, a).$$

It preserves positive responses and sets negative responses to zero. In contrast, $|a|$ preserves the magnitude of both signs. Thus applying ReLU to a signed edge response discards one edge orientation; taking its absolute value treats the two orientations alike. The fitted network in the companion uses tanh, not ReLU; these are alternative activation choices.



Computed deterministic results: training loss decreases over 160 epochs; test accuracy is 50% for the constant baseline and 100% for both the fixed-filter rule and the small trained MLP. All 96 test images come from the same synthetic generator family.

6 Baselines and generalization

Slides for this section

A constant classifier and a fixed-filter rule establish what can be achieved without training a neural network. The supplied synthetic experiment is simple enough that both a suitable fixed rule and the small fitted network can classify its held-out generator samples perfectly. This result does not establish robustness to new acquisition processes, rotations, artifacts, or biological images.

The comparison holds the training/test split fixed and evaluates

constant rule, fixed spatial filter, trained network.

A higher-capacity model is useful only if it improves the relevant task.

Distribution shift

A test set drawn from the same generator checks interpolation within that experiment.

A changed generator tests a different claim. Specify which image transformations and noise changes are included before evaluating robustness.

Exercises

N5-1. Trace a tiny network

For one example, x=[2, 1]. Use first-layer weight rows [1, -1] and [0.5, 1], bias [-1, 0], then ReLU. The output weights are [2, -1], output bias -2, and sigmoid gives the class 1 probability.
a Compute the two first-layer values and the ReLU activations.

b Compute the logit and approximate probability.

c If the label is 1, what is the output-bias gradient for this one example, and which direction does gradient descent move it?

N5-2. Shapes and shared parameters

A batch contains 5 single-channel 8×8 images. Apply two 3×3 filters at stride 1 without padding.
a State the input, filter and output shapes using the lesson's axis order.

b How many learned parameters would two such filters have if each had one learned bias? How many are learned in the fixed-filter demonstration?

c What shape remains after averaging absolute responses across both spatial axes? What distinction does ReLU make instead?

N5-3. Evaluate before enlarging the claim

On 80 held-out examples, a model has confusion counts [[36, 4], [10, 30]], with true classes in rows and predicted classes in columns. Both true classes contain 40 examples.
a Compute accuracy and class 1 recall.

b Compare with a constant baseline and state one limitation of accuracy alone.

c A student tries ten hidden widths and reports the best score on this same test set. What must change before a final unbiased evaluation?

N5-4. Check a computation and its limits

For one parameter w, a loss calculation gives L (w+0.001)=0.420310 and L (w-0.001)=0.419710. The coded derivative is 0.30001.

a Compute the centered finite-difference derivative and its absolute discrepancy from the coded value.

b For w=0.7 and learning rate 0.05, what update follows from the finite-difference slope?

c A constant-intensity image gives zero response to the two derivative filters. Does that make it a valid vertical or horizontal example?

References

Original Python Lecture 2. PDF pp. 30–33 and 41: array products and shapes.
https://zanahmad.com/files/scientific-python/Lecture_2__Intro_to_Python.pdf

Original derivation and executable checks in this lesson. Sections 2–5 and demo.py: forward map, stable binary loss, gradient check and fixed-filter calculation.

Python source materials: Phillip Kerger (2023). Course adaptation and additional exposition: Zan Ahmad.