Losses and generalization

D.A. Forsyth

University of Illinois at Urbana Champaign

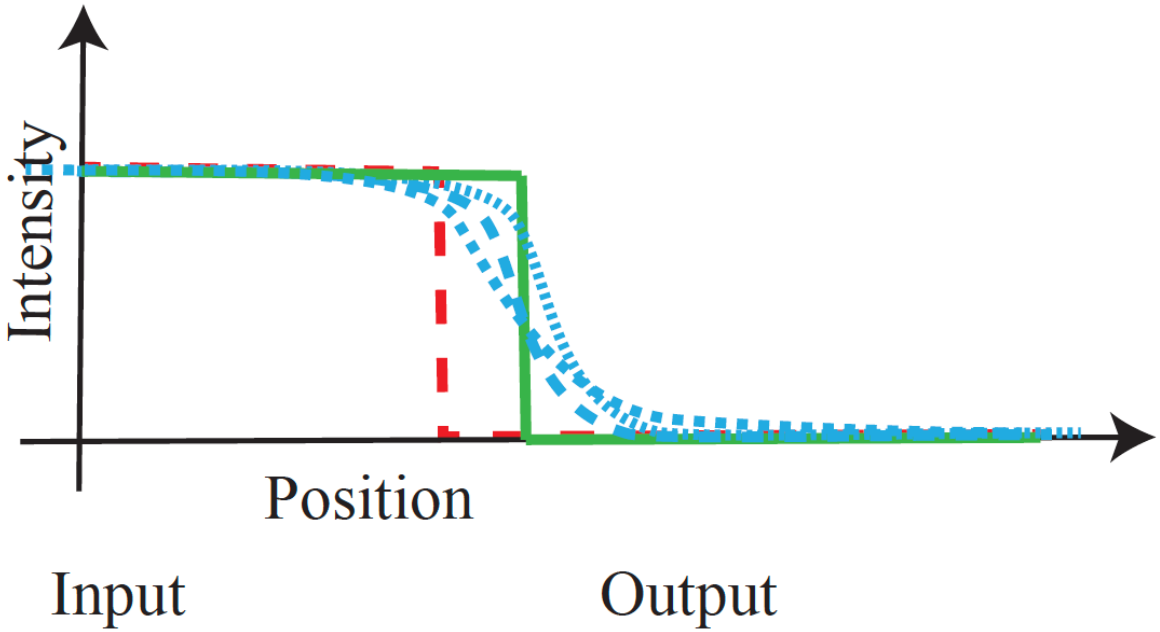
Losses – the L2 loss

Loss functions typically evaluate *residuals* – the difference between what the system provides and ground truth. The *SSD loss* compares a reconstructed training image $\mathcal{R}$ to the ground truth $\mathcal{G}$ by

$$\mathcal{C}_{L2}(\mathcal{R}, \mathcal{G}) = \sum_{ij} \Delta_{ij}^2,$$

where $\Delta_{ij} = \mathcal{R}_{ij} - \mathcal{G}_{ij}$ is the *residual*. This is the square of the L2 norm of Δ, and is sometimes (rather disreputably) referred to as an *L2 loss*. This might seem a

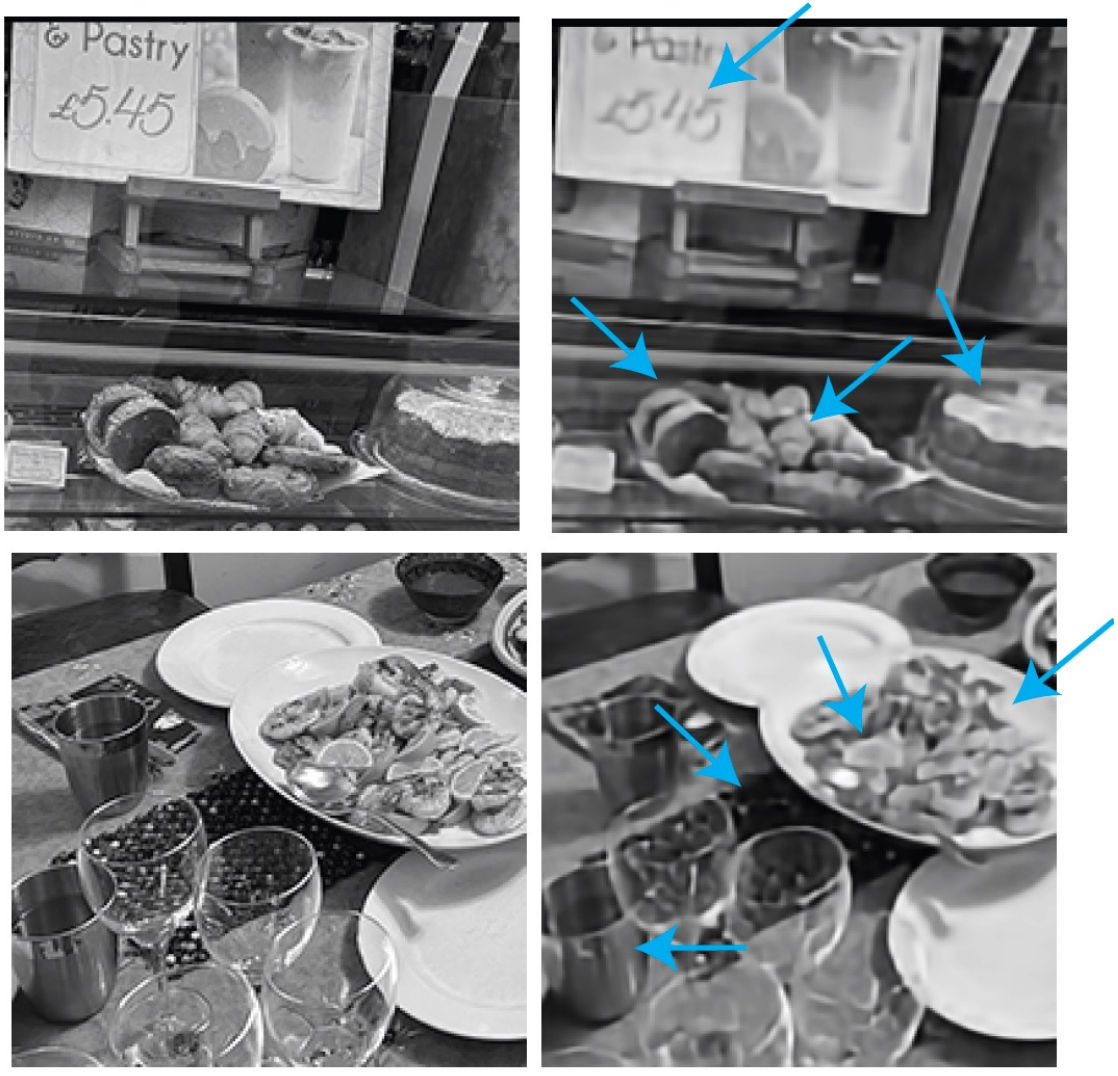
L2 loss creates blur



- Sharp edge in the wrong place (red) is expensive
- Compared to blurry edge in about the right place (blue)

Input

Output



L1 loss

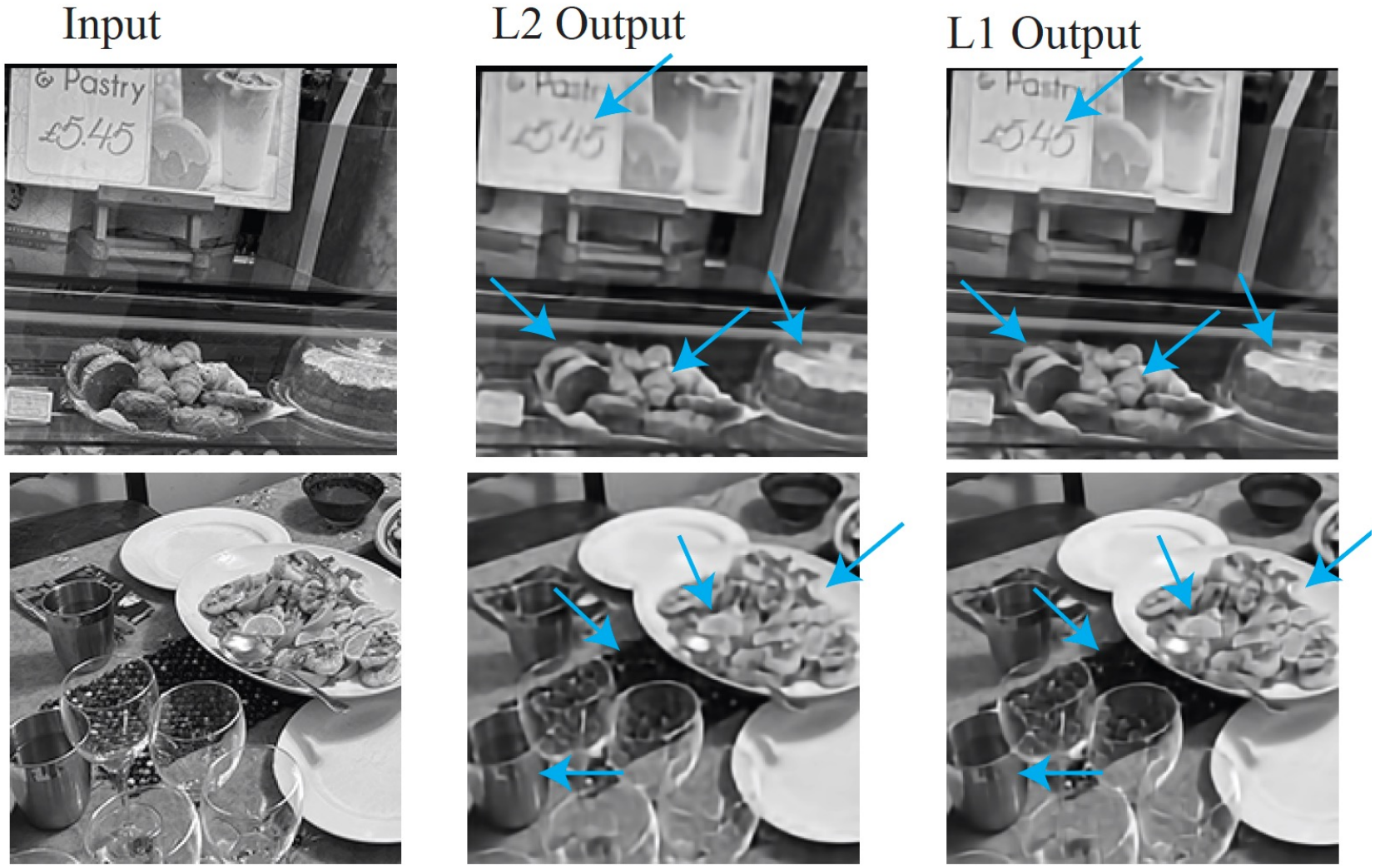
the gradient to have zeros, assuming the optimization process can cope. Using an L1 term, written

$$\mathcal{C}_{L1}(\mathcal{R},\mathcal{G}) = \sum_{ij} |\Delta_{ij}|$$

will tend to encourage the residual to have zeros in it, and will tend to discourage blurring (Figure **??**).

- Idea:
  - penalize absolute value of residual
- We saw the L1 norm in denoising
- Square of a small number is very small; absolute value of small number isn't
- Tends to discourage blurring

L1 loss



Losses and Gradients

- Notice that L1 loss, L2 loss DON'T:
  - Force values to be non-negative
  - Force values to be less than 1
- (VERY BAD) Idea:

Here is an example of a bad loss. The *indicator function* is a function that tests its argument against a condition, then reports 1 if the condition is true and zero otherwise. For example,

$$\mathbb{I}_{[x<0]}(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases}$$

is 1 when $x < 0$ and 0 otherwise. Note some redundancy here; the condition usually means it is obvious what the argument is, so it is quite usual to write $\mathbb{I}_{[x<0]}$ rather than $\mathbb{I}_{[x<0]}(x)$. The following (BAD) choice of loss could be intended to force an output to be non-negative:

$$\mathcal{C}_{\text{bad}}(\mathcal{I}) = \sum_{ij} \mathbb{I}_{[\mathcal{I}_{ij}<0]}$$

Losses and Gradients

- Bad, because it supplies no gradient
  - Pixel is +ve: loss and gradient are zero
  - Pixel is –ve: loss is 1, gradient is zero – no information about how to change filters
- CF L1 loss:
  - Pixel value too large: gradient pushes it down
  - Pixel value too small: gradient pushes it up
  - Pixel value just right: non-differentiability doesn't matter
    - never happens
    - choose -1<= gradient <=1: everything works fine

General ideas: Cheating

- SGD is a very effective search
  - Astonishing, but true
  - It might find a solution you don’t expect and don’t want
- Example:
  - train an autoencoder with (clean, clean) pairs
  - it will cheat, and pass on pixels
  - even if it has a complicated architecture
  - symptoms:
    - can’t denoise
    - meaningless image representation

General ideas: Cheating

- Cheating example above won’t denoise
- It is important to train representation to denoise images
- (Noisy, clean) pairs will produce something useful
- (clean, clean) pairs won’t!

General ideas: Generalization

- No interest in denoising training images
- We want to denoise new images

- Options:
  - Data augmentation: make training dataset look bigger
  - Regularization: make it hard to choose filters that are specialized

General ideas: Data Augmentation

- Applies to many learned systems

- For now:
  - A left/right flipped image is still an image
  - An up/down flipped image is still an image
  - An image crop is still an image
  - Etc

- When forming a batch, randomly
  - crop, flip, etc. images

General ideas: Regularization

- Prefer small coefficients to large coeffs.
  - A filter with large coeffs that works well on training data might produce an unexpected large response on new data
- Discourage filters with large coeffs by penalizing loss
  - (cost of error on batch) + scale*(penalty for large coeffs)
- Known as weight decay
  - API will do this for you if you ask

General ideas: Regularization

- How to choose scale?

- Train for many different choices
- Evaluate each on held out data to choose
- Now re-train using the best value
- Evaluate the result on new dataset

General ideas: dropout

Another regularization strategy is *dropout*, where one randomly replaces elements of a data block with zeros during training. This is intended to advantage filters that are robust to error. Dropout will tend to disadvantage a filter that relies too strongly on one input, because that input might be dropped out. Some housekeeping is required to implement dropout properly, because the filter sees a "smaller" input in training (where some inputs might be zeroed) than in test. A good API will have a dropout implementation that takes care of this, and I leave the topic to the manual of your API. Further strategies involve discouraging large values in data blocks (*normalization*) and are dealt with in Section 18.2.3.

- This is a form of regularization

Things to think about...

17.7. You want to ensure that an autoencoder produces a non-negative number at every location. Section 17.4.2 says that

$$\sum_{ij} \frac{1}{1 + e^{a\mathcal{I}_{ij}}}$$

would be a bad choice of loss for $a > 0$ and large. What happens if $a > 0$ and small? what happens if $a < 0$?

17.8. Section 17.4.1 says: "Recall from Section 7.2.4 that using an L1 norm as a penalty for the gradient tends to cause the gradient to have zeros, assuming the optimization process can cope **exercises** ." Explain; do you expect gradient descent on an L1 loss produce zeros?