# Computer vision

## Image intensities.

- A monochrome image can be represented as a matrix of intensity values $I(x,y)$, where we typically use 256 discrete values (8 bits) to represent the intensity values.
- The size of the matrix is typically 320×240 (QVGA), 640×480 (VGA), 1280×720 (HD), 3840×2160 (4K) etc.
- We can represent colour by stacking 3 matrices (RGB)

## Nuisance factors in image data.

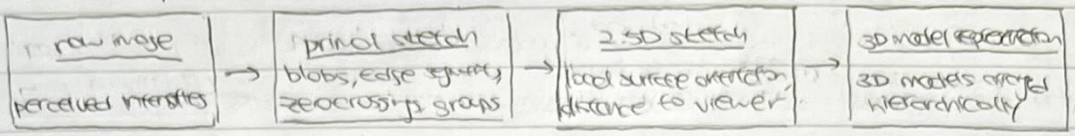
- If a pt on an object is visible in view, the intensity $I(x,y)$ is a function of many geometric and photometric variables:
  - ↳ The position and orientation of the camera.
  - ↳ The geometry of the scene (3D shape + layout)
  - ↳ The nature and distribution of light sources.
  - ↳ The reflectance properties of the surfaces (specular / (diffuse reflection, albedo))
  - ↳ The properties of the camera lens and sensor array.
- In practice, the pt. may only be partially visible / appearance affected by occlusion

## Data reduction

- W/ current computers, we need to discard most data from the camera before an attempt can be made at real-time interpretation.
  (raw image data ~ 10GB/s, generic salient features ~ 100 kB/s)
- In the process of reducing raw image data to generic features, we aim to:
  - ↳ Allow the image to be discarded, and have all subsequent processes done on the features themselves → dramatically reduces the amt of data.
  - ↳ Preserve useful info in images (e.g. 2D shape of objects in scene)
  - ↳ Discard redundant info in the images (e.g. lighting conditions)
  - ↳ Be as generic as possible → can process the same way for a wide range of applications

Computer vision as hierarchical processing.

- Marr's hierarchy is as follows:




level of difficulty: lower-level features → mid-level features → high-level features.

- For deep learning, the model automatically learns all the layers of the hierarchy.

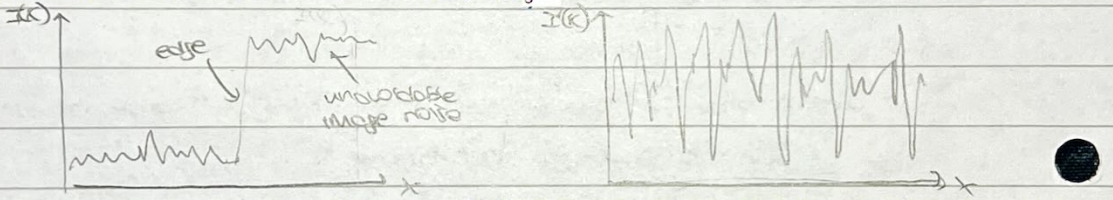
Image structure.

- Patches of an image can be a featureless region, edge or corner.
- The featureless region is characterised by a smooth variation of intensities [0D]
  The patch containing the edge reveals an intensity discontinuity in one dir. [1D]
  The patch containing the corner reveals an intensity discontinuity in two dirs [2D].
- An edge/corner representation imparts a desirable invariance to lighting — the intensity discontinuities are likely to be prominent, regardless of the lighting conditions.

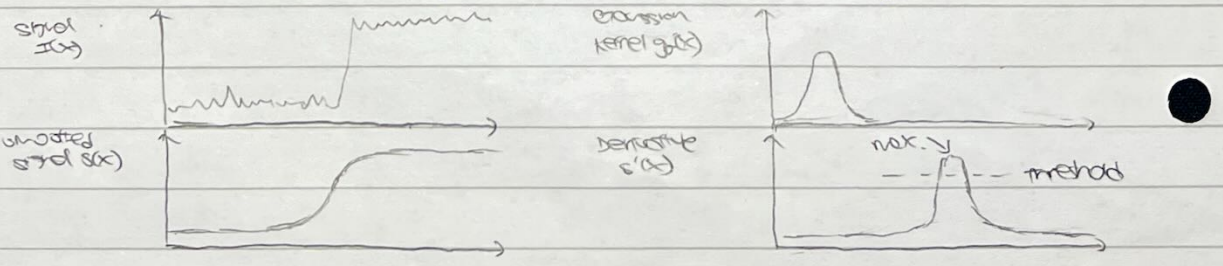
## Feature extraction

### 1D edge detection

- consider the signal I(x) w/ an obvious edge

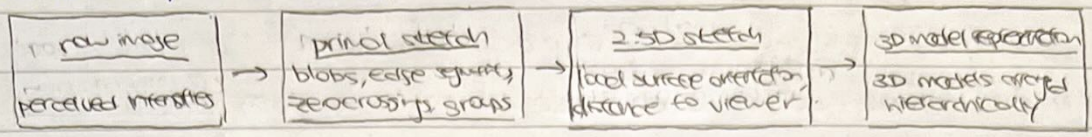



- An intuitive approach to edge detection might be to look for max/min. in I(x). However, our simple strategy was defeated by high freq. noise (amplified by differentiation).
- Therefore, all edge detectors start by smoothing the signal to suppress noise → use Gaussian filter.
- A broad overview of the 1D edge detection algorithm is as follows:
  ↳ 1) convolve the signal I(x) w/ a Gaussian kernel $g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{x^2}{2\sigma^2}\right)$ to produce smooth S(x).
  ↳ 2) Compute S'(x), the derivative of S(x).
  ↳ 3) Find the max/min. of S'(x).
  ↳ 4) use thresholding on the magnitude of the extrema to mark edges.



Computer vision as hierarchical processing.

- Marr's hierarchy is as follows:




level of abstraction: lower-level features → mid-level features → high-level features.

- For deep learning, the model automatically learns all the layers of the hierarchy.

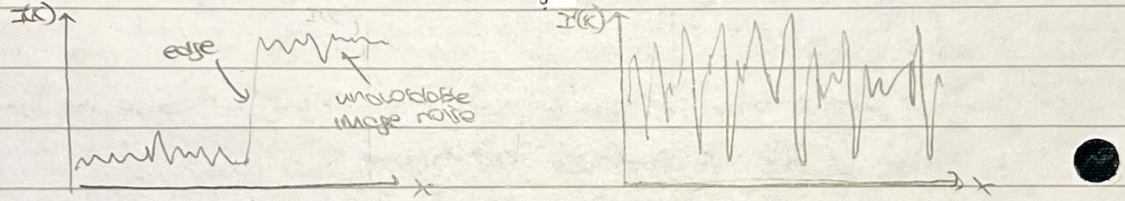
Image structure.

- Patches of an image can be a featureless region, edge or corner.
- The featureless region is characterised by a smooth variation of intensities [0D]
  The patch containing the edge reveals an intensity discontinuity in one dir. [1D]
  the patch containing the corner reveals an intensity discontinuity in two dirs [2D].
- An edge/corner representation imparts a desirable invariance to lighting — the intensity discontinuities are likely to be prominent, regardless of the lighting conditions.

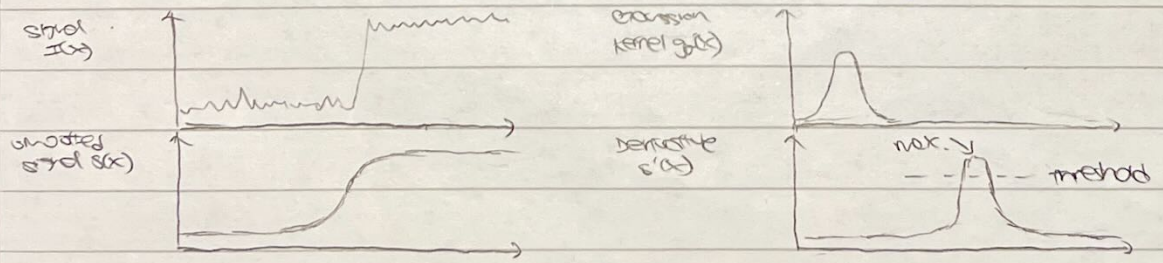
## Feature extraction

### 1D edge detection

- consider the signal $I(x)$ w/ an obvious edge




- An intuitive approach to edge detection might be to look for max/min. in $I(x)$. However, our simple strategy was defeated by high freq. noise (amplified by differentiation).
- Therefore, all edge detection start by smoothing the signal to suppress noise → use Gaussian filter.
- A broad overview of the 1D edge detection algorithm is as follows:
  - 1) convolve the signal $I(x)$ w/ a Gaussian kernel $g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{x^2}{2\sigma^2})$ to produce smooth $S(x)$.
  - 2) Compute $S'(x)$, the derivative of $S(x)$.
  - 3) Find the max/min. of $S'(x)$.
  - 4) use thresholding on the magnitude of the extrema to mark edges.



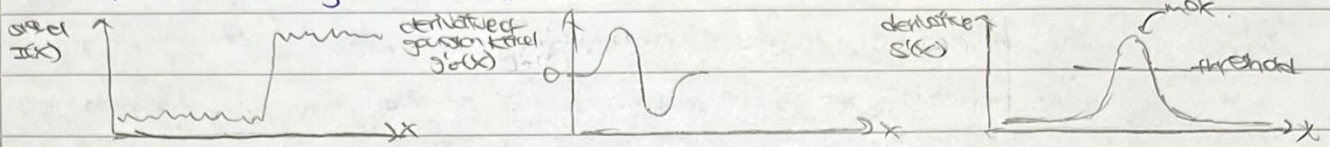
Computational trick for 1D edge detection

- The smoothing in step 1 / differentiation in step 2 is performed by a 1D convolution → ff seems edge detection req. two computationally expensive convolutions.
- However, using the derivative thm of convolution,

$$s'(x) = \frac{d}{dx}[g_\sigma(x) * I(x)] = g'_\sigma(x) * I(x)$$

we can compute $s'(x)$ w/ just a single convolution → considerable computational saving.

- A faster variant of the edge detection algorithm is as follows:
  - 1) Convolve the signal $I(x)$ w/ a derivative of Gaussian kernel $g'_\sigma(x)$ to produce $s'(x)$ directly.
  - 2) Find the max/min of $s'(x)$
  - 3) Use thresholding on the magnitude of the extrema to mark edges.

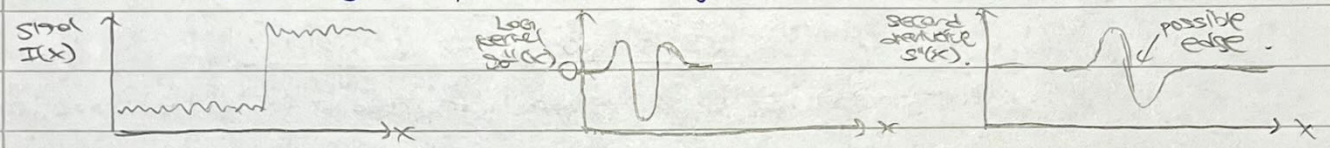



- Finding max./min. of $s'(x)$ is the same as looking for the zero-crossings of $s''(x)$. Applying the derivative thm. of convolution for a second time,

$$s''(x) = \frac{d}{dx}[g'_\sigma(x) * I(x)] = g''_\sigma(x) * I(x)$$

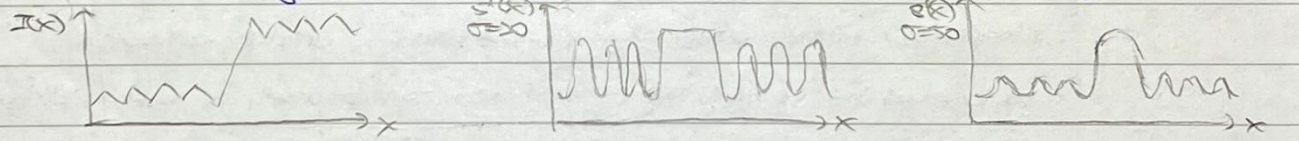
where $g''_\sigma(x)$ is the Laplacian of Gaussian ("LoG kernel").

- The fastest variant of the edge detection algorithm involves convolving the signal $I(x)$ w/ the LoG kernel $g''_\sigma(x)$ to get $s''(x)$, where zero-crossings of $s''(x)$ mark possible edges.




Scale of 1D edge detection.

- The choice of $\sigma$ in the Gaussian kernel used determines the scale of edges we want to detect.
- Using a small $\sigma$ brings out many edges. As $\sigma$ increases, the signal is smoothed more and more, and only the central edge survives.
- Consider the following signal $I(x)$ and the derivatives of the smoothed signal $s'(x)$ using $\sigma=20$ and $\sigma=50$.




- The amt of smoothing controls the scale at which we analyse the image.
- There is no right or wrong choice of $\sigma$ for the Gaussian kernel — it depends on the scale we're interested in.
- Modest smoothing (small $\sigma$) brings out edges at fine scale; More smoothing (large $\sigma$) finds edges at larger scales, suppressing finer detail

* Fine scale edge detection is particularly sensitive to noise.

## 2D edge detection

- The 1D edge detection scheme can be extended to work in 2D as follows:
  - ↳ 1) Smoothing: we smooth the image $I(x,y)$ by convolving w/ a 2D Gaussian $G_\sigma(x,y) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$

$$S(x,y) = G_\sigma(x,y) * I(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G_\sigma(u,v) I(x-u, y-v)\,du\,dv$$

  - ↳ 2) Gradients: we compute the gradient of the smoothed image at every pixel.

$$\nabla S = \nabla(G_\sigma * I) = \begin{bmatrix} \frac{\partial}{\partial x}(G_\sigma * I) \\ \frac{\partial}{\partial y}(G_\sigma * I) \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix}$$

  - ↳ 3) Non-maxima suppression (NMS): edge elements (edgels) are placed at locations where $|\nabla S|$ is greater than local values of $|\nabla S|$ in the dir. $\pm\nabla S$ → ensure all edgels are located at ridge pts of surface $|\nabla S|$
  - ↳ 4) Threshold: the edgels are thresholded so only those w/ $|\nabla S|$ above a certain value are retained.
- The output is a list of edgel positions, each w/ strength $|\nabla S|$ and orientation $\frac{\nabla S}{|\nabla S|}$
- The above is the Canny edge detection algorithm, which is a directional edge finder.
- An alternative is the Marr & Hildreth edge detection algorithm, which finds the zero-crossings of $\nabla^2 G_\sigma * I$. The Marr-Hildreth operator is isotropic (not directional).

## Implementation details for edge detection

① Truncated summations

- In practice, the image and filter kernels are discrete quantities and the convolutions are performed as a truncated sum (since we cannot do an infinite sum)

$$S(x,y) = \sum_{u=-n}^{n}\sum_{v=-n}^{n} G_\sigma(u,v)\, I(x-u, y-v)$$

(filter kernel: $2n+1$ pixel)

- For acceptable accuracy, kernels are generally truncated s.t. the discarded samples are less than 1% of the peak value (e.g. for $\sigma = 1$, we req. $2n+1 = 7$)

② 2D convolutions

- 2D convolutions are computationally expensive, but can be decomposed into 2× 1D convolutions

$$G_\sigma(x,y) * I(x,y) = g_\sigma(x) * [g_\sigma(y) * I(x,y)]$$

- The computational saving is $\frac{(2n+1)^2}{2(2n+1)} = \frac{1}{2}(2n+1)$

③ Differentiation via convolution

- Differentiation of the smoothed image is implemented w/ discrete convolution.
- Taylor expanding the derivative of $S(x,y)$ wrt $x$ to first order, we have a finite diff. approximation.

$$\frac{\partial S}{\partial x} = \frac{S(x+1,y) - S(x-1,y)}{2}$$

- This can be calculated by convolving the rows of smoothed image storing $S(x,y)$ w/ the kernel

| $1/2$ | $0$ | $-1/2$ |
|---|---|---|

* When convolving, we flip the kernel and sum the element-wise products under each kernel position

$$\begin{bmatrix} S(x+1,y) & S(x,y) & S(x+1,y) \end{bmatrix} * \begin{bmatrix} 1/2 & 0 & -1/2 \end{bmatrix} = S(x+1,y)\left(-\tfrac{1}{2}\right) + S(x,y)\cdot(0) + S(x+1,y)\left(\tfrac{1}{2}\right)$$
$$= \frac{S(x+1,y) - S(x-1,y)}{2}$$

## Cross correlation

- The cross correlation function btwn two diff. signals is defined as the measure of similarity or coherence btwn a signal and the time delayed version of another signal.

$$\mathrm{CC}(x,y) = \sum_{u=-n}^{n}\sum_{v=-n}^{n} P(u,v)\, I(x+u, y+v)$$

- The normalised cross correlation measures how well an image patch $P(u,v)$ matches portions of an image $I(x,y)$, that have the same size as the patch.

$$C(x,y) = \frac{\sum_{u=-n}^{n}\sum_{v=-n}^{n}(P(u,v)-\bar{P})(I(x+u,y+v)-\bar{I}_{x,y})}{\sqrt{\sum_{u=-n}^{n}\sum_{v=-n}^{n}(P(u,v)-\bar{P})^2}\sqrt{\sum_{u=-n}^{n}\sum_{v=-n}^{n}(I(x+u,y+v)-\bar{I}_{x,y})^2}}$$

covariance of $I, I$; variance of $P$; variance of $I$

where $\bar{P}$ / $\bar{I}_{xy}$ is the mean pixel value of the patch (image under the patch).

- Normalising cross correlation to $[-1,1]$ increases its robustness to illumination changes.
- Cross-correlation is v. similar to convolution, but we do not flip the kernel/signal.

## Sum of squared differences and cross correlation

- The sum of squared differences (SSD)/ Euclidean distance intuitively is a metric for comparing how similar two patches are. For a patch $P(u,v)$ and image $I(x,y)$ of size $(2n+1)\times(2n+1)$,

$$\mathrm{SSD}(x,y) = \sum_{u=-n}^{n}\sum_{v=-n}^{n}\left(P(u,v) - I(x+u,y+v)\right)^2$$

Expanding the expression, we get

$$\mathrm{SSD}(x,y) = \sum_{u=-n}^{n}\sum_{v=-n}^{n} P(u,v)^2 - 2P(u,v)\,I(x+u,y+v) + I(x+u,y+v)^2$$

- The first term in $\mathrm{SSD}(x,y)$, $P(u,v)^2$ is const wrt $x,y$.

In natural images, pixel values often vary smoothly → approx last term in $\mathrm{SSD}(x,y)$, $I(x+u,y+v)^2$ by a const.

$$\therefore \mathrm{SSD}(x,y) \approx -2\cdot \mathrm{CC}(x,y) + \text{const}$$

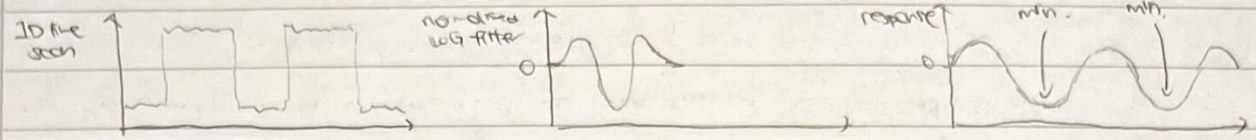
→ greater correlation implies greater similarity (smaller Euclidean distance).

## Corner detection

- The motion of an edge is rendered ambiguous by the aperture problem — when viewing a moving edge, it is only possible to measure the motion normal to the edge only
- To measure image motion in 2D completely, we can look at corner features
- A patch w/ a well-defined peak in its autocorrelation (self cross correlation) can be classified as a corner (featureless region has no peaks; edge produces a ridge)

Blob detection.

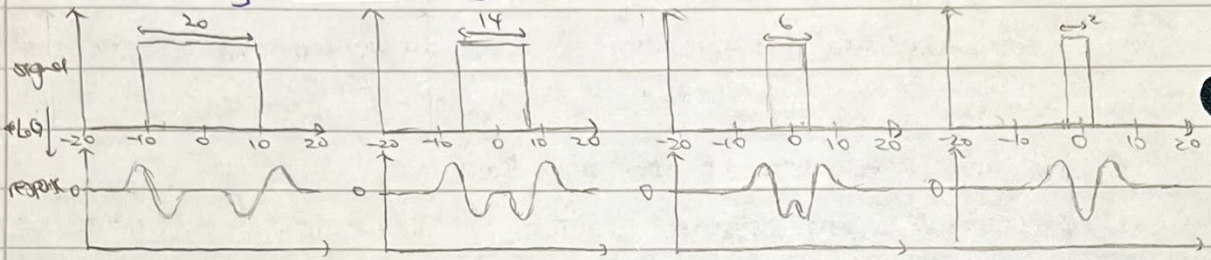
- A blob is an area of uniform/similar intensity in the image → useful for predicting scale
- Blobs can be detected w/ the normalised LoG filter.




Despite a noisy signal, the minima of the response from the scale normalised LoG at the correct scale, σ, localise the centres of bright blobs on a dark background perfectly.

* Dark blobs on a bright background produce maxima

- Consider applying a LoG filter ($\sigma=1$) to a box function of diff. widths.



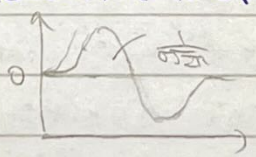

Convolving w/ the LoG filter gives "ripples" w/ zero-crossings at the position of the edges. When the scales match, we get a superposition of the ripples to give a strong we response

→ therefore bright blobs on a dark background give a minima.

- The size of the blob detected depends on the σ-value of the LoG filter used. As σ increases, larger and longer image features are detected.
- Each time, the blob detector will fire on the centre of the blob in question → useful for extracting texture from inside the object / fixing location of object in the scene.

The scale normalised LoG filter.

- A scale normalised LoG filter is req. for the blob detector.
- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases.



To produce the same response across diff. σ values, we must multiply the Gaussian derivative by σ.

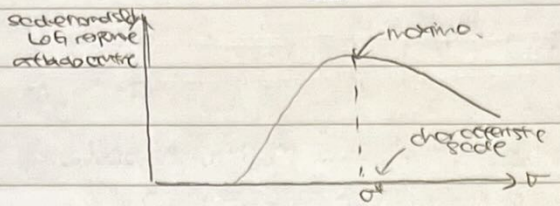
- Since the Laplacian is the second derivative of the Gaussian, we must multiply by $\sigma^2$ to scale normalise.

$$\nabla^2_{norm} G = \sigma^2 \nabla^2 G$$

- If we don't scale normalise, it will be difficult to detect max/min. for large σ.

## characteristic scale

- Blobs have a range of scales over which they will be detected.
- The scale normalised LoG as recorded at a particular location is a smooth function over scale, w/ definite peaks / troughs — the max/min. occur at the centre of blobs
- The characteristic scale of a blob is the scale corresponding to the max. of the detector response at the blob centre (ideal scale for a key pt.) → defines the size of the blob.




- Since the function is continuous, to find the exact pt./scale of the blob, a set of discrete scales are sampled (using an image pyramid) and the max is found via interpolation

## Scale space

- To estimate the scale of a structure, we look at diff. resolutions of an image (LPF at diff. scales), and selecting the scale that gives the strongest response
- There is an infinite no. of possible resolutions for an image, which together form a 3D function of intensity over location and scale — the scale space of the image $S(x,y,\sigma)$.
- We can calculate $S(x,y,\sigma)$ by convolving the original image $I(x,y)$ w/ Gaussians of diff. scales. $\sigma$, thus the scale space can be written as

$$S(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

where $G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$

- It is impractical to examine all possible resolutions → sample the space by choosing particular resolutions
- We produce a discrete set of LPF images by smoothing w/ Gaussians w/ a scale satisfying.

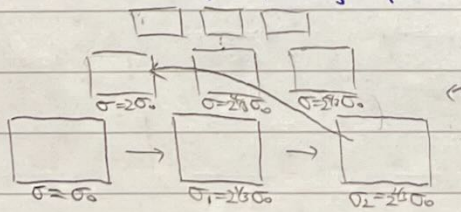
$$\sigma_i = 2^{\frac{1}{s}}\sigma_0$$

so it doubles after $s$ intervals (go up an octave). The $s$ images in each octave are spaced logarithmically w/ the scale of neighbouring images satisfying.

$$\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$$

- Each time the scale doubles in scale space, the blurring removes sufficient high freq. information that we can subsample (downsample) the image by a factor of 2 w/o losing information (Nyquist sampling thm.)

$s=3$




- Blurring small images is computationally cheaper as we process fewer pixels and the use of w. large kernels to compute responses at large scales (since $\sigma\uparrow \rightarrow 2\sigma+1\uparrow$)

- Even within octave, blurring w/ larger Gaussian kernels is computationally expensive → use incremental blurs. We can use the reproducing property of the Gaussian.

$$G(\sigma_1) * G(\sigma_2) = G\left(\sqrt{\sigma_1^2+\sigma_2^2}\right)$$

- We want each incremental blur to satisfy $\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$ → find $\sigma_{ki}$ s.t. $G(\sigma_{i+1}) = G(\sigma_i) * G(\sigma_{ki})$

Reproducing property:

$$\sigma_{i+1} = \sqrt{\sigma_i^2 + \sigma_{ki}^2}$$

$$\sigma_{ki}^2 = \sigma_{i+1}^2 - \sigma_i^2$$

$\sigma_{i+1} = 2^{\frac{1}{s}}\sigma_i$:

$$\sigma_{ki}^2 = \left(2^{\frac{2}{s}} - 1\right)\sigma_i^2$$

$$\sigma_{ki} = \sigma_i\sqrt{2^{\frac{2}{s}} - 1}. \quad \text{[incremental blur size]}$$

- This gives a distinct and small incremental Gaussian filters ($\sigma_{ki}$), which need only be computed once. They can be reused in each subsequent octave (subsampled images, larger scales) → no large convolutions req.
  - when subsampling by a factor of two, we effectively blur w/ a larger kernel
- The difference of Gaussians "DoG" filter approximates the scale normalised LoG filter

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G(x,y,\sigma)$$

- In a system which uses a scale space pyramid, we can calculate DoGs by finding the difference btwn neighbouring images of same dim. in the image pyramid.

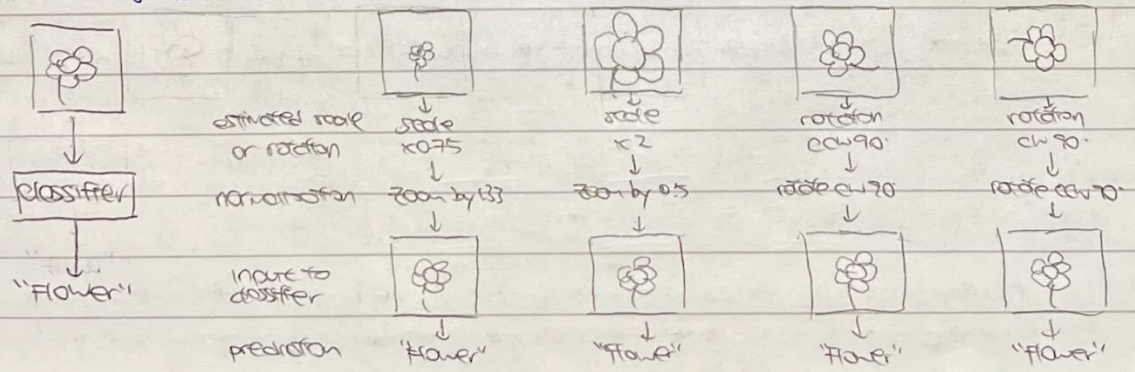
## Matching and correspondence

Matching and correspondence.

- The ability to match image structures btwn images is an important primitive in computer vision
- Successful matching enables us to establish correspondences across views — find pairs of regions for which "this bit" in one image matches "that bit" in another.

Invariance.

- To work in the real world, we want our models to be invariant to certain properties of objects.
- An object recognition model that has invariance in scale/rotation gives the same output for diff. scales/rotations of input
- Suppose the model works at one scale/rotation. If we can estimate the scale/rotation of the object, we can normalise it to achieve scale/rotation invariance.



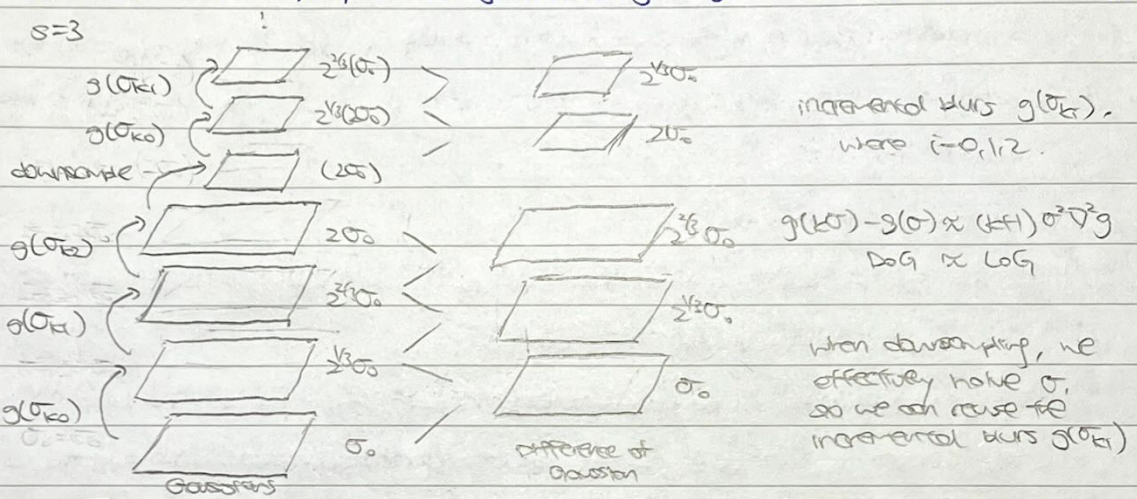
Keypoints and descriptors

- Scale and rotation invariance are v. useful, but they are not enough on their own:
  - ↳ we need robustness wrt additional variance to factors like partial occlusion, and changes in 3D view pt and illumination
  - ↳ too much invariance is bad — we also need to create features that are distinctive.
- To achieve invariance + distinctiveness for robust matching, we can use key pts + descriptors:
  - ↳ key pts enable us to estimate (and thus normalise and achieve invariance to) scale and rotation
  - ↳ Descriptors enhance distinctiveness, while supporting partial invariance to changes in 3D view, occlusion and illumination

RKE

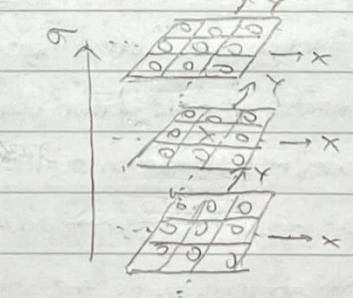
Scale-invariant keypoint detection.

- Keypoint locations (blob centres) are found by first computing an approximation for the LoG filter using the DoG filter.
- This is done efficiently by subtracting neighbouring images of same dimension in the image pyramid.




The location of the local max./min. of DoG response (in image position and over scale) gives the keypoint location and characteristic scale.

- To find the local extrema, we do a local search of 26 neighbour responses to determine if a pixel is a blob centre + of its characteristic scale. (check if it has the largest/smallest value)



## Raw pixel intensity patches

- We could use raw pixel intensity patches as descriptors.
- The simplest way to "describe" a patch of N pixels in an image is to store the N intensity values P[i].
- We can then compare patches directly using cross-correlation to find a match

$$CC(P_1, P_2) = \sum_{i=1}^{N} P_1[i]\, P_2[i]$$

- This raw form of description is not robust to changes in lighting

## Zero-normalised intensity patches

- Brightness changes are changes in the mean brightness value → giving the intensity values a zero mean, they become rel. immune to brightness changes.

$$M = \frac{1}{N}\sum_{x,y} I(x,y) \qquad Z(x,y) = I(x,y) - M$$

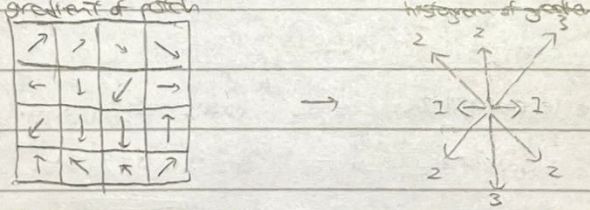
- Contrast changes are changes in the variance of distribution of brightness abt the mean → normalising by dividing each value by the s.d. can deal w/ the contrast changes.

$$\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2 \qquad Z_N(x,y) = \frac{Z(x,y)}{\sigma}$$

- Zero-normalised patches often can be accurately matched using simple cross correlation.
- ↓ The size of the descriptor grows w/ the size of the patch, and thus can be quite big.

## Histogram of oriented gradients

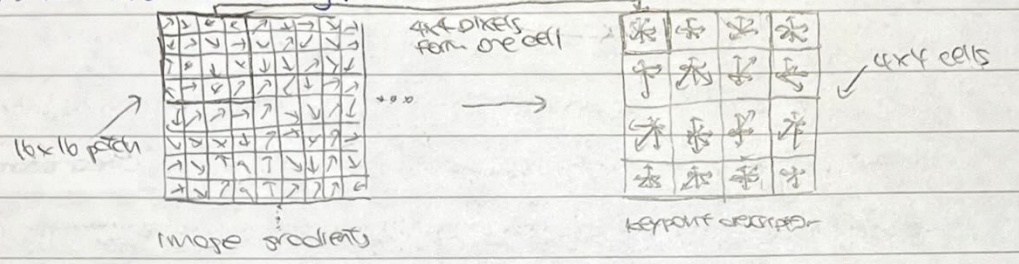
- Considering the gradient of each pixel in the patch, each will have its own distinct orientation/dir. and size/strength (gradient magnitude).
- The pixel gradients can be binned together into a "histogram of oriented gradients" (HOG).
- The histogram is built using gradients (robust to contrast/brightness changes), and can be detected at diff. scales, and also incorporate discriminative orientation data. → descriptor + estimate orientation




- To find the dominant orientation, we build a histogram (typically 36 bins covering 360°) of all the edge orientations weighted by their gradient magnitudes in the neighbourhood of the keypt.
- Before computing the gradients, we weight the magnitudes w/ a 2D Gaussian ($\sigma$=1.5*) centred on the keypt. to ensure the pixels closer to the keypt have greater influence.
- The highest peak in the histogram will approx. the dominant orientation → use this orientation to estimate the rotation of a keypt → normalise to achieve rotation invariance.
- A better estimate can be found through interpolation
- If there is no clear max, then the keypt is given several dominant orientations (several copies of the keypt w/ diff orientations are used)

The SIFT keypoint descriptor.

- SIFT stands for scale-invariant feature transform. It uses a collection of oriented histograms to create a robust and descriptive representation of a patch.
- The N×N patch (typically, N=16) is extracted at the scale of the keypt, and its gradient orientations are stored rel. to the dominant orientation of the keypt → scale+rotation invariant.
- The N×N patch is split into c cells (typically, N cells) and the dir. are binned into a histogram weighted by their magnitude.
- A Gaussian window ($\sigma = 0.5N$) is applied at the centre of the patch so the inner pixels (closer to kept.) have a greater weighting, and min. the influence of partial occlusions.




- If the bins are centred on d dir. (typically 8), in each of c cells (typically 16), the resulting descriptor is a dxc vector (typically 128).
- Once the dxc vector has been extracted it is L2 normalised (Euclidean) to provide invariance to gradient magnitude change
- To min. the effects of non-affine lighting changes, the values are truncated s.t. all the values in the unit vector are less than 0.2, then renormalising.
- By dividing the patch into cells, a particular gradient can move and to some degree within the descriptor window and still contribute to the same directional histogram

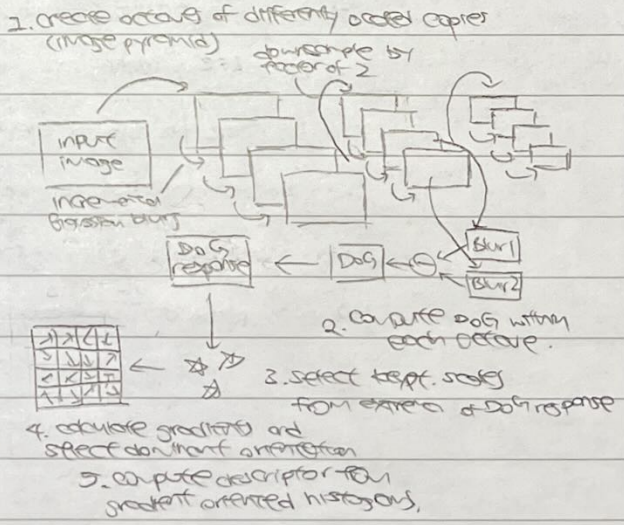
Keypoint and descriptor framework overview

① Finding keypoint scale and location

- We first normalise for scale in the image (by sampling image intensities at the appropriate level of the image pyramid) → scale invariance.

② Finding keypoint orientation

- Then normalise for orientation (by finding the dominant orientation using a HoG)



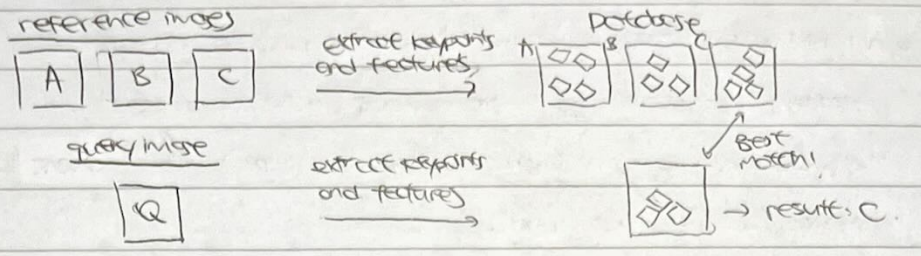

③ Computing the feature descriptor

- Calculate the feature descriptor using SIFT

& SIFT struggles w/ large changes in viewpt and background clutter at object boundaries

## Matching features over multiple views

- We can use our design of keypts + descriptors to build a system to recognise a target object (specified by a reference image) from another viewpt (query image)




- A good match is usually defined as one which is a small distance away in feature-descriptor space ($k=128$ for SIFT) as measured by Euclidean distance $E(X,Y)$,

$$E(X,Y) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

- One way to solve the correspondence problem is to search through all keypoint descriptors in the database image for the best match of a query feature
- Data structures can be used to organise data so it is more efficient to store, access, search (kd trees good for nearest-neighbour retrieval).

## Image features

### Image feature

- A texture is a visual pattern on an infinite 2D plane which, at some scale, has a stationary distribution.
  Eg: banded, dotted, chequered...
- We can characterise feature through its response to a filter bank.
- An example of a filter bank (w/ 48 filters) would be:
  - 8 LoG filters + 4 Gaussian filters at diff. scales to provide non-oriented responses. (blob)
  - 36 oriented filters at 6 angles, 3 scales and 2 phases. to provide oriented responses (bar, edge)
- The descriptor is simply the concatenated responses of all the filters in the filter bank at a pixel
- We can define textons as the prototypes that arise from clustering the responses of a filter bank.
- Since filter banks respond to basic image features (blobs, edges, bars), they are inherently robust to many kinds of illumination change in an image.

# Deep learning

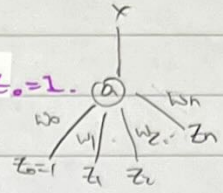
## single neuron and logistic classification

### A single neuron

- An artificial neuron computes a nonlinear function of inputs.

$$x = \sigma(a) = \frac{1}{1+\exp(-a)} \quad x \in (0,1)$$ (sigmoid)

$$a = w_0 + \sum_{d=1}^{D} w_d z_d = \sum_{d=0}^{D} w_d z_d$$ (inputs, bias, weights)




where $x$ is the output of the neuron and $a$ is the activity of the neuron.

and $w_d$ is the weight, $w_0$ is the bias (combined into the summation by setting $z_0=1$)

- On a plot in the input space $z_i$, we have activity contours ($a = \underline{w}^T \underline{z} = \text{const}$), which act as boundaries,

↳ $\frac{\underline{w}}{|\underline{w}|}$ sets the direction of the boundary ↳ $|\underline{w}|$ sets the steepness of the boundary

### Training a single neuron

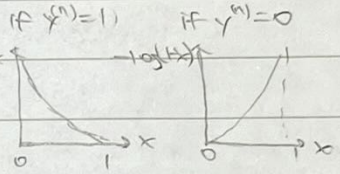
- The training data includes inputs $\{\underline{z}^{(n)}\}_{n=1}^{N}$ and class labels $\{y^{(n)}\}$.

- We want to tweak the values of the weights and biases $\underline{w}$ so the neuron outputs:

$$x(\underline{z}^{(n)}; \underline{w}) \approx 1 \text{ for } y^{(n)} = 1 \qquad x(\underline{z}^{(n)}; \underline{w}) \approx 0 \text{ for } y^{(n)} = 0.$$

- To achieve this, we set up an objective function $G(\underline{w})$

$$G(\underline{w}) = -\sum_n \left[ y^{(n)} \log x^{(n)} + (1-y^{(n)}) \log(1-x^{(n)}) \right] \geq 0.$$ (binary cross)




which is the rel. entropy btwn $x$ and $y^{(n)}$.

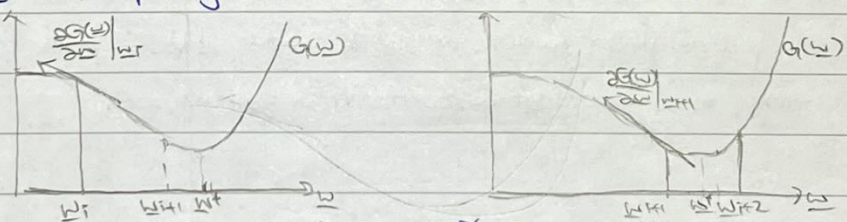
We want to choose the weights and biases to min. the network's surprise abt training data, ie.

$$\underline{w}^* = \underset{\underline{w}}{\operatorname{argmin}}\, G(\underline{w})$$

- There is no analytic soln to this optimisation problem.

### Gradient descent

- We can find $\underline{w}^*$ iteratively using gradient descent. Consider a 1D representation:




- We initialise the weight $\underline{w}_i$ and compute the gradient. The gradient points away from the local min. So we increase the weight $\underline{w}_i \to \underline{w}_{i+1}$ by taking a small step in opposite dir. to the gradient.

$$\underline{w}_{i+1} = \underline{w}_i - \eta \frac{\partial G(\underline{w})}{\partial \underline{w}}\Big|_{\underline{w}_i}$$ (length of step)

where $\eta$ is the learning rate.

- If the learning rate $\eta$ is too large, we may move away from optimum and can then diverge

If the learning rate $\eta$ is too small, it may take a long time to converge to the optimum.

Computing gradient descent.

- Given we want to find $\underline{w}^*$ to minimise $G(\underline{w})$, where

$$G(\underline{w}) = -\sum_n \left[ y^{(n)} \log x^{(n)} + (1-y^{(n)}) \log(1-x^{(n)}) \right], \quad \underline{w}^* = \operatorname*{argmin}_{\underline{w}} G(\underline{w})$$

we compute $\frac{dG(\underline{w})}{d\underline{w}}$ for gradient descent. (prediction error)

$$\frac{d}{d\underline{w}} G(\underline{w}) = \sum_n \frac{dG(\underline{w})}{dx^{(n)}} \frac{dx^{(n)}}{d\underline{w}} = -\sum_n (y^{(n)} - x^{(n)}) \underline{z}^{(n)}$$ (feature)

$$\therefore \quad \underline{w} \leftarrow \underline{w} - \eta \frac{d}{d\underline{w}} G(\underline{w})$$

- To compute $\frac{d}{d\underline{w}} G(\underline{w})$, first consider $\frac{dG(\underline{w})}{dx^{(n)}}$, then $\frac{dx^{(n)}}{d\underline{w}}$. (drop superscript to avoid clutter)

$$\frac{dG(\underline{w})}{dx} = \frac{d}{dx}\left[-(y \log x + (1-y)\log(1-x))\right] = -\left(y \cdot \frac{1}{x} - (1-y)\frac{1}{1-x}\right) = -\frac{y(1-x) - (1-y)x}{x(1-x)} = -\frac{y-x}{x(1-x)}$$

$$\frac{dx}{d\underline{w}} = \frac{d}{d\underline{w}}\left[\frac{1}{1+\exp(-\sum_k w_k z_k)}\right] = -\frac{\exp(-\sum_k w_k z_k)\cdot(-\underline{z})}{(1+\exp(-\sum_k w_k z_k))^2} = \underline{z}\, x^2\left(\frac{1}{x} - 1\right) = \underline{z}\, x(1-x).$$

$$\therefore \frac{dG(\underline{w})}{d\underline{w}} = \sum_n -\frac{y^{(n)} - x^{(n)}}{x^{(n)}(1-x^{(n)})} \underline{z}^{(n)} x^{(n)}(1-x^{(n)})$$

$$= \sum_n (y^{(n)} - x^{(n)}) \underline{z}^{(n)}$$

Overfitting and weight decay

- Overfitting is when we can perfectly fit training data, but generalise to test data poorly.

- This can be avoided by including a regulariser $E(\underline{w})$, where

$$E(\underline{w}) = \frac{1}{2}\sum_i w_i^2$$

which penalise magnitude of the weights and discourages the network from using extreme weights.

- So now we choose weights and biases $\underline{w}$ s.t.

$$\underline{w}^* = \operatorname*{argmin}_{\underline{w}} M(\underline{w}) = \operatorname*{argmin}_{\underline{w}} [G(\underline{w}) + \alpha E(\underline{w})]$$

and the derivative used in gradient descent $\frac{dM(\underline{w})}{d\underline{w}}$ becomes

$$\frac{d}{d\underline{w}} M(\underline{w}) = -\sum_n (y^{(n)} - x^{(n)}) \underline{z}^{(n)} + \alpha \underline{w}$$ (weight decay → shrinks weight towards 0.)

where $\alpha$ is a hyperparameter that tunes the penalty of weights.

- Adding a regulariser prevents the use of extreme weights → prevent overconfidence (though it would still be overconfident w/ "outlier" data)

Probabilistic interpretation of the single neuron

- We can interpret the output of the network as a probability

$$x(\underline{z}^{(n)}; \underline{w}) = p(y^{(n)} = 1 \mid \underline{z}^{(n)}, \underline{w})$$

- We can choose two diff. fittings:

↳ Maximum-likelihood learning (ML): choose setting of $\underline{w}$ that makes observed data most probable

$$\underline{w}_{ML} = \operatorname*{argmax}_{\underline{w}} p\left(\{y^{(n)}\}_{n=1}^N \mid \{\underline{z}^{(n)}\}_{n=1}^N, \underline{w}\right)$$

↳ Maximum a posteriori (MAP): choosing setting of $\underline{w}$ that is most probable given the data and prior knowledge.

$$\underline{w}_{MAP} = \operatorname*{argmax}_{\underline{w}} p\left(\underline{w} \mid \{\underline{z}^{(n)}\}_{n=1}^N, \{y^{(n)}\}_{n=1}^N\right)$$

① Maximum-likelihood learning : $\underline{w}_{ML} = \arg\max_{\underline{w}} p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w})$

$p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w}) = \prod_{n=1}^N p(y^{(n)} \mid z^{(n)}, \underline{w})$ , $p(y^{(n)} \mid z^{(n)}, \underline{w}) = \begin{cases} x(z^{(n)}, \underline{w}) & y^{(n)}=1 \\ 1-x(z^{(n)}, \underline{w}) & y^{(n)}=0 \end{cases}$

$= \prod_{n=1}^N x(z^{(n)}, \underline{w})^{y^{(n)}} \cdot (1-x(z^{(n)}, \underline{w}))^{(1-y^{(n)})}$ Algebra trick

$\underline{w}_{ML} = \arg\max_{\underline{w}} p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w}) = \arg\max_{\underline{w}} \log p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w})$ monotonicity of log preserve location of optimum

$= \arg\max_{\underline{w}} \sum_{n=1}^N y^{(n)} \log x(z^{(n)}, \underline{w}) + (1-y^{(n)}) \log(1-x(z^{(n)}, \underline{w})) = \arg\min_{\underline{w}} (-G(\underline{w}))$

log-likelihood

$\underline{w}_{ML} = \arg\min_{\underline{w}} (G(\underline{w}))$

② Maximum a posteriori : $\underline{w}_{MAP} = \arg\max_{\underline{w}} p(\underline{w} \mid \{z^{(n)}\}_{n=1}^N, \{y^{(n)}\}_{n=1}^N)$

Using Bayes rule, $p(A|B,C) = \frac{p(A|C)p(B|A,C)}{p(B|C)}$, we have $A = \underline{w}$, $B = \{y^{(n)}\}_{n=1}^N$, $C = \{z^{(n)}\}_{n=1}^N$

$\therefore p(\underline{w} \mid \{y^{(n)}\}_{n=1}^N, \{z^{(n)}\}_{n=1}^N) = \frac{p(\underline{w} \mid \{z^{(n)}\}_{n=1}^N)\, p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w})}{p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N)}$ prior, likelihood, normalizing const.

Using a zero mean, independent Gaussian prior $p(w \mid \{z\}_{n=1}^N) = \frac{1}{(2\pi\sigma_w^2)^{D/2}} \exp\left(-\frac{1}{2\sigma_w^2} \sum_{d=1}^D w_d^2\right)$

$\underline{w}_{MAP} = \arg\max_{\underline{w}} p(\underline{w} \mid \{z^{(n)}\}_{n=1}^N, \{y^{(n)}\}_{n=1}^N) = \arg\max_{\underline{w}} \log p(\underline{w}, \{z^{(n)}\}_{n=1}^N, \{y^{(n)}\}_{n=1}^N)$ monotonicity of log preserves location of optimum

$= \arg\max_{\underline{w}} \left[\log p(w \mid \{z^{(n)}\}_{n=1}^N) + \log p(\{y^{(n)}\}_{n=1}^N \mid \{z^{(n)}\}_{n=1}^N, \underline{w})\right]$

$= \arg\min_{\underline{w}} \left[-\frac{D}{2}\log(2\pi\sigma_w^2) - \frac{1}{2\sigma_w^2}\sum_{d=1}^D w_d^2 - G(\underline{w})\right]$ (doesn't depend on $\underline{w}$)

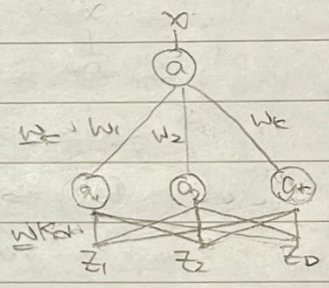
$\underline{w}_{MAP} = \arg\min_{\underline{w}} \left[G(\underline{w}) + \alpha E(\underline{w})\right]$ where $\alpha = \frac{1}{\sigma_w^2}$

## Neural networks and multilayer perceptrons

### Single hidden layer neural networks

- We can add a single hidden layer btwn the input and outputs to get nonlinear classification contours (in the input space plot)

| | |
|---|---|
| output of network | $x(a) = \frac{1}{1+\exp(-a)}$ |
| activity of neuron output | $a = \sum_{k=1}^K w_k x_k$ |
| hidden neuron input | $x_k(a_k) = \frac{1}{1+\exp(-a_k)}$ |
| hidden neuron activities | $a_k = \sum_{d=1}^D w_{kd} z_d$ |
| inputs | $\{z_i\}_{i=1}^D$ |




- the dimensionality of input/output weights $w/\underline{w}$ are

↳ input : $\dim(\underline{w}) = K \times D$ ↳ output : $\dim(\underline{w}) = K$.

### Backpropagation

- For neural networks w/ hidden layers, we still have the objective function $G$ and regularizer $E$

$G(\underline{w}) = -\sum_n \left[y^{(n)} \log x^{(n)} + (1-y^{(n)}) \log(1-x^{(n)})\right]$ $\quad E(\underline{w}, \underline{w}) = \frac{1}{2}\sum_i w_i^2 + \frac{1}{2}\sum_{ij} w_{ij}^2$

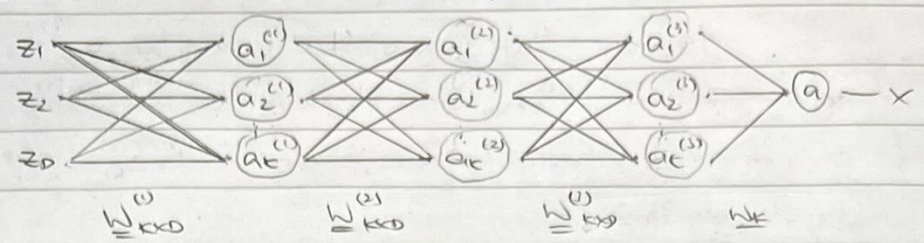
$\rightarrow \{\underline{w}, \underline{w}^*\} = \arg\min_{\underline{w}, \underline{w}} M(\underline{w}, \underline{w}) = \arg\min_{\underline{w}, \underline{w}} \left[G(\underline{w}, \underline{w}) + \alpha E(\underline{w}, \underline{w})\right]$

- Now the derivative used in gradient descent $\frac{dM}{dw_{ij}}$ involves the chain rule — backpropagation

$\frac{dM(\underline{w})}{dw_{ij}} = \sum_n \frac{dG(\underline{w})}{dx^{(n)}} \cdot \frac{dx^{(n)}}{dw_{ij}} = \sum_n \frac{dG(\underline{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dw_{ij}} = \sum_n \frac{dG(\underline{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dw_{ij}} = \sum_n \frac{dG(\underline{w})}{dx^{(n)}} \frac{dx^{(n)}}{da^{(n)}} \frac{da^{(n)}}{dw_{ij}}$

## Multi-layer perceptrons (MLP)

- We can add more hidden layers btwn the inputs and outputs — multi-layer perceptrons.




## Convolutional neural networks

Problem of MLPs.

- MLPs are hard to train — over-fitting, slow to converge.
  MLPs have a large memory footprint → limit complexity of trained network.
- For a MLP w/ 3 hidden layers, each w/ D nodes, the no. of parameters req. $|\theta|$ is

$$|\theta| = 3D^2 + D.$$

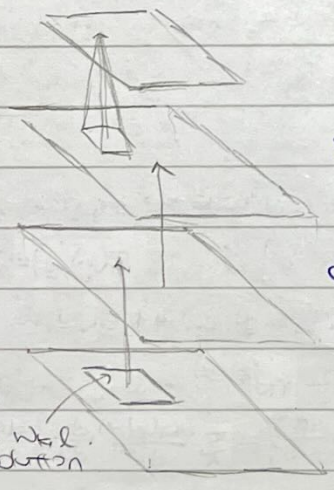
(hidden layers; final neuron)

(For a small $32\times32$ image, $|\theta| = 3\times10^6$).

- Convolutional nets reduce the no. of parameters w/o compromising network complexity

## Convolutional neural networks (CNN)

- The key ideas behind CNNs are as follows:
  - ↳ image statistics are translation invariant — build this translation invariance into the model by tying lots of the weights together in the network → ↓ # of parameters
  - ↳ low-level features learned to be local — build this into the model by allowing only local connectivity → ↓ # of parameters
  - ↳ high-level features learned to be coarser — build this into the model by subsampling more and more up the hierarchy → ↓ # of parameters.
- The building block of a CNN is as follows



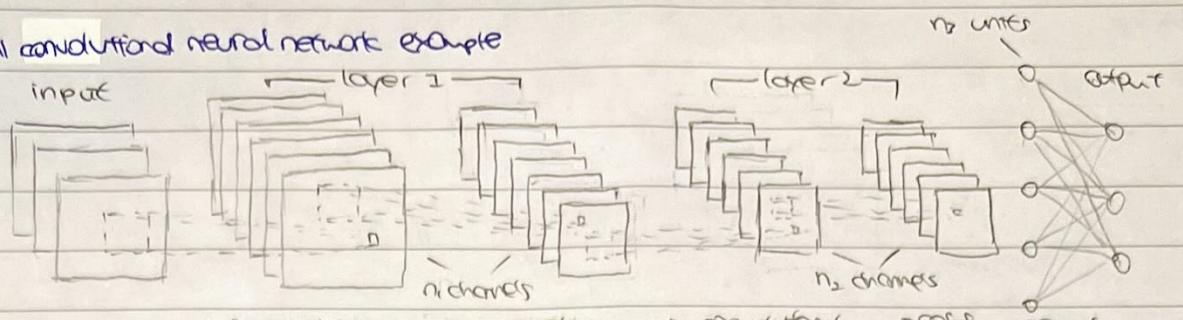

pooling stage: $x_{ij} = \max_{|k|<r,|l|<r} y_{i+k,j-l}$ (mean/subsample)

non-linear stage: $y_{i,j} = f(a_{ij})$ where $f(a) = \text{ReLU}(a), \sigma(a)$

convolutional stage: $a_{ij} = \sum_{k,l} w_{kl} z_{i-k,j-l}$ (only parameters)

input image: $z_{ij}$

## Full convolutional neural network example




* hyperparameters are the convolution/pooling filter size, no. of channels, stride.

## Training – stochastic gradient descent

- The output of a CNN can be interpreted as a class label probability

$$x^{(n)} = x(z^{(n)}; w) = p(y^{(n)}=1 \mid z^{(n)}, w)$$

where $x^{(n)}$ is a more complex function of the inputs $z^{(n)}$ and certain set of parameters $w$

- We train by optimising the cross entropy (w/ regularisation/weight decay) as before.

$$G(w) = \sum_{n=1}^{N} \left(y^{(n)} \log x^{(n)} + (1-y^{(n)}) \log(1-x^{(n)})\right) = \sum_{n=1}^{N} \ell(y^{(n)}, x^{(n)}) \quad \text{(loss)}$$

$$\frac{dG(w)}{dw} = \sum_{n=1}^{N} \frac{d\ell(y^{(n)}, x^{(n)})}{dw} \qquad w \leftarrow w - \eta\left(\frac{dG(w)}{dw} - \alpha w\right).$$

However, this will take a long time to accumulate objective/derivative over all $N$ training data pts.

- Since we often only need to see a small no. of images before we get a good idea of the gradient direction, at each step of gradient descent, we can randomly select $M \leq N$ data pts from the training set $\{y^{(n)}, z^{(n)}\}_{n=1}^{M}$, and estimate the objective/gradient from these

batch size.

stochastic gradient descent

$$G(w) = \sum_{n=1}^{N} \ell(y^{(n)}, z^{(n)}) \approx \frac{N}{M} \sum_{n=1}^{M} \ell(y^{(n)}, z^{(n)})$$

$$\frac{dG(w)}{dw} = \sum_{n=1}^{N} \frac{d\ell(y^{(n)}, z^{(n)})}{dw} \approx \frac{N}{M} \sum_{n=1}^{M} \frac{d\ell(y^{(n)}, z^{(n)})}{dw}$$

- In practice, we don't choose these samples at random, we randomly shuffle the training set and go through it in chunks

## Additional tips and tricks for training.

- Take care to initialise scale of weights correctly (large weights → output saturated → gradient tends to zero)
- Use non-saturating hidden units (ReLu) rather than sigmoid units.
- Applying data augmentation improves performance substantially (inc. shifted, rotated, mirrored, cropped, locally distorted version of images as training data)