CPSC 340: Machine Learning and Data Mining

Convolutions

Fall 2022

Last Time: Feature Engineering

- We discussed feature engineering:
  - Designing a set of features to achieve good performance on a problem.
- We discussed various issues:
  - Feature aggregation/discretization to address coupon counting.
  - Feature scaling to address features of different scales.
  - Non-linear transforms to make relationships more linear.
- We started discussing feature engineering on text data:
  - Bag of words:
    - Loses a LOT of information.
    - But let’s us learn fast if word order isn’t that relevant.
  - Trigrams (“sets of 3 adjacent words”):
    - Captures local context of a word.
    - But requires collecting a lot of coupons: $3^{(\text{number of words})}$.

Text Example 3: Part of Speech (POS) Tagging

- Consider problem of finding the verb in a sentence:
  - “The 340 students jumped at the chance to hear about POS features.”
- Part of speech (POS) tagging is the problem of labeling all words.
  - >40 common syntactic POS tags.
  - Current systems have ~97% accuracy on standard (“clean”) test sets.
  - You can achieve this by applying a “word-level” classifier to each word.
    - That independently classifies each word with one of the 40 tags.
- What features of a word should we use for POS tagging?

POS Features

- Regularized multi-class logistic regression with these features gives ~97% accuracy:
  - Categorical features whose domain is all words (“lexical” features):
    - The word (e.g., “jumped” is usually a verb).
    - The previous word (e.g., “he” hit vs. “a” hit).
    - The previous previous word.
    - The next word.
    - The next next word.
  - Categorical features whose domain is combinations of letters (“stem” features):
    - Prefix of length 1 (“what letter does the word start with?”)
    - Prefix of length 2.
    - Prefix of length 3.
    - Prefix of length 4 (“does it start with JUMP?”)
    - Suffix of length 1.
    - Suffix of length 2.
    - Suffix of length 3 (“does it end in ING?”)
    - Suffix of length 4.
  - Binary features (“shape” features):
    - Does word contain a number?
    - Does word contain a capital?
    - Does word contain a hyphen?
- Total number of features: ~2 million (same accuracy with ~10 thousand using L1-regularization).

Ordinal Features

- Categorical features with an ordering are called ordinal features.

Rating
Bad
Very Good
Good
Good
Very Bad
Good
Medium

Rating
2
5
4
4
1
4
3

- If using decision trees, makes sense to replace with numbers.
  – Captures ordering between the ratings.
  – A rule like (rating ≥ 3) means (rating ≥ Good), which make sense.

Ordinal Features

- With linear models, "convert to number" assumes ratings are equally spaced.
  - "Bad" and "Medium" distance is similar to "Good" and "Very Good" distance.
- One alternative that preserves ordering with binary features:

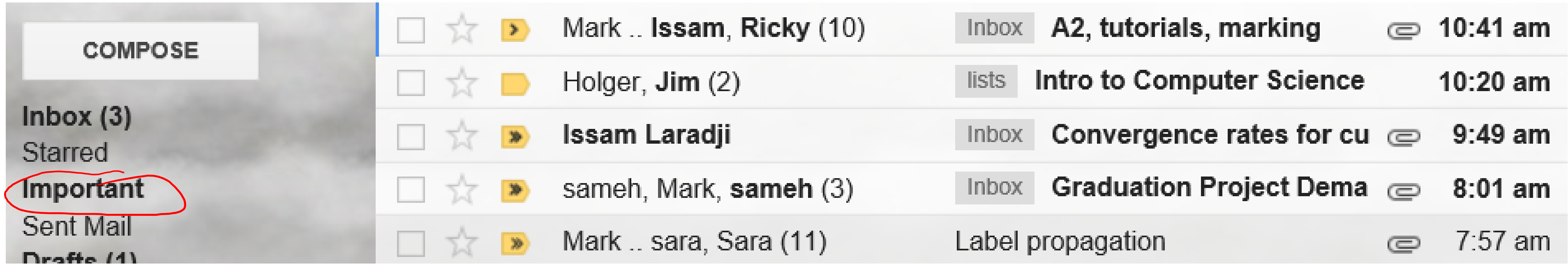
Rating
Bad
Very Good
Good
Good
Very Bad
Good
Medium

≥ Bad	≥ Medium	≥ Good	Very Good
1	0	0	0
1	1	1	1
1	1	1	0
1	1	1	0
0	0	0	0
1	1	1	0
1	1	0	0

- Regression weight w_{medium} represents:
  - "How much medium changes prediction over bad".
- Bonus slides discuss "cyclic" features like "time of day".

Next Topic: Personalized Features

Motivation: "Personalized" Important E-mails



- Features: bag of words, trigrams, regular expressions, and so on.

- There might be some "globally" important messages:
  - "This is your mother, something terrible happened, give me a call ASAP."

- But your "important" message may be unimportant to others.
  - Similar for spam: "spam" for one user could be "not spam" for another.

“Global” and “Local” Features

- Consider the following weird feature transformation:

“340”
1
1
1
0
1

$\Rightarrow$

“340” (any user)	“340” (user?)
1	User 1
1	User 1
1	User 2
0	<no “340”>
1	User 3

- First feature: did “340” appear in this e-mail?
- Second feature: if “340” appeared in this e-mail, who was it addressed to?

- First feature will increase/decrease importance of “340” for every user (including new users).
- Second (categorical feature) increases/decreases importance of “340” for a specific user.
  – Lets us learn more about specific users where we have a lot of data

“Global” and “Local” Features

- Recall we usually represent categorical features using “1 of k” binaries:

“340”
1
1
1
0
1

$\Rightarrow$

“340” (any user)	“340” (user = 1)	“340” (user = 2)
1	1	0
1	1	0
1	0	1
0	0	0
1	0	0

- First feature “moves the line up” for all users.
- Second feature “moves the line up” when the e-mail is to user 1.
- Third feature “moves the line up” when the e-mail is to user 2.

The Big Global/Local Feature Table for E-mails

- Each row is one e-mail (there are lots of rows):

$X = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$ $y = \begin{bmatrix} \text{"important"} \\ \text{"not important"} \\ \vdots \end{bmatrix}$

"global" features: shared by all users

"local" features for user "1": set to 0 for all other users.

"local" features for user "2"

We only need to store the user ID and list of non-zero features.

Predicting Importance of E-mail For New User

- Consider a new user:
  - We start out with no information about them.
  - So we use global features to predict what is important to a generic user.

$$\hat{y}_i = \text{sign}(\underbrace{w_g^T x_{ig}}_{\text{features/weights shared across users.}})$$

  - Weights on local/user features are initialized to zero.
- With more data, update global features and user's local features:
  - Local features make prediction *personalized*.

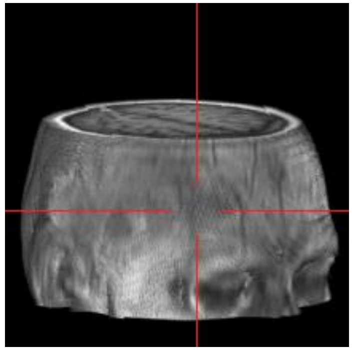
$$\hat{y}_i = \text{sign}(w_g^T x_{ig} + \underbrace{w_u^T x_{iu}}_{\text{features/weights specific to user.}})$$

  - What is important to *this* user?
- G-mail system: classification with logistic regression.
  - Trained with a variant of stochastic gradient descent (later).

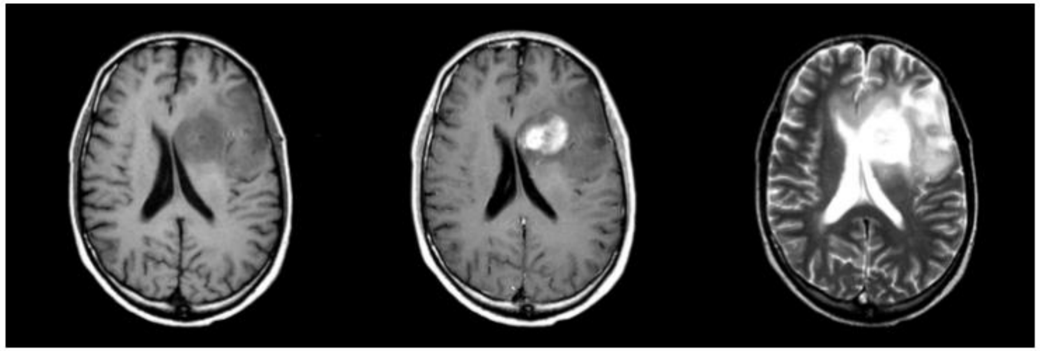
Next Topic: Convolutions

Motivation: Automatic Brain Tumor Segmentation

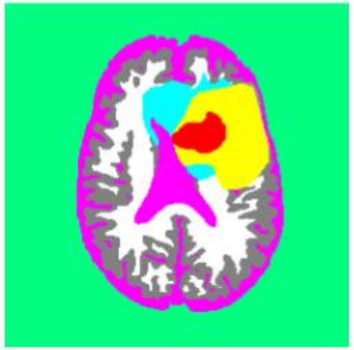
- Task: labeling tumors and normal tissue in multi-modal MRI data.



Input:



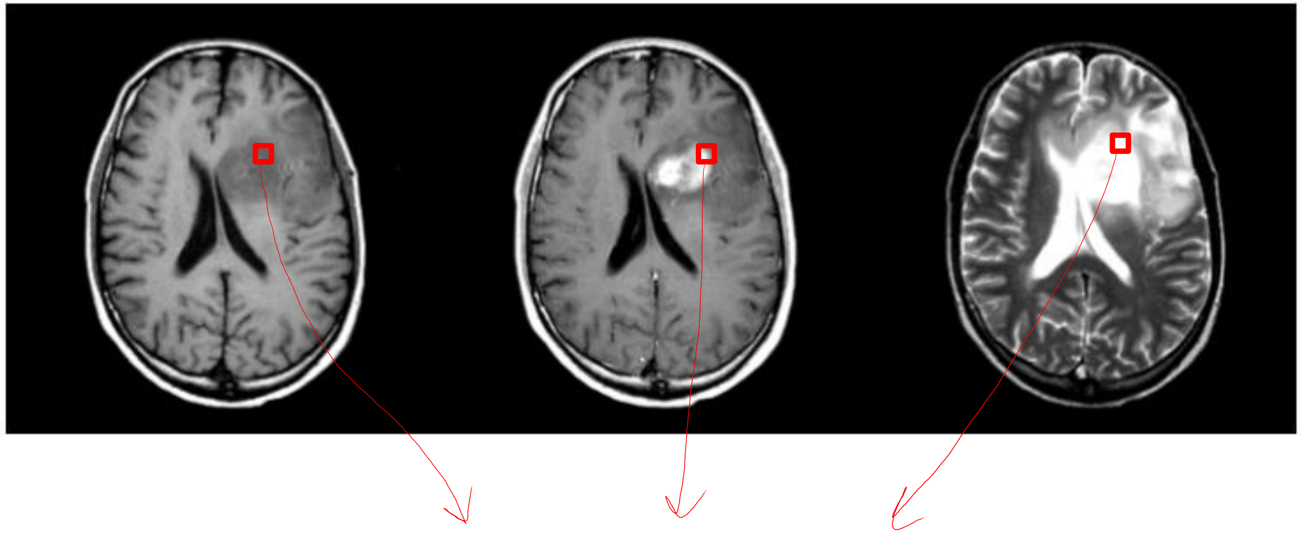
Output:



- Applications:
  - Radiation therapy target planning, quantifying treatment responses.
  - Mining growth patterns, image-guided surgery.
- Challenges:
  - Variety of tumor appearances, similarity to normal tissue.
  - Grumbly scientist to me in 2003: “you are never going to solve this problem.”

Naïve Voxel-Level Classifier

- We could treat classifying a voxel as supervised learning:
  - Standard representation of image: each pixel gets “intensity” between 0 and 255.

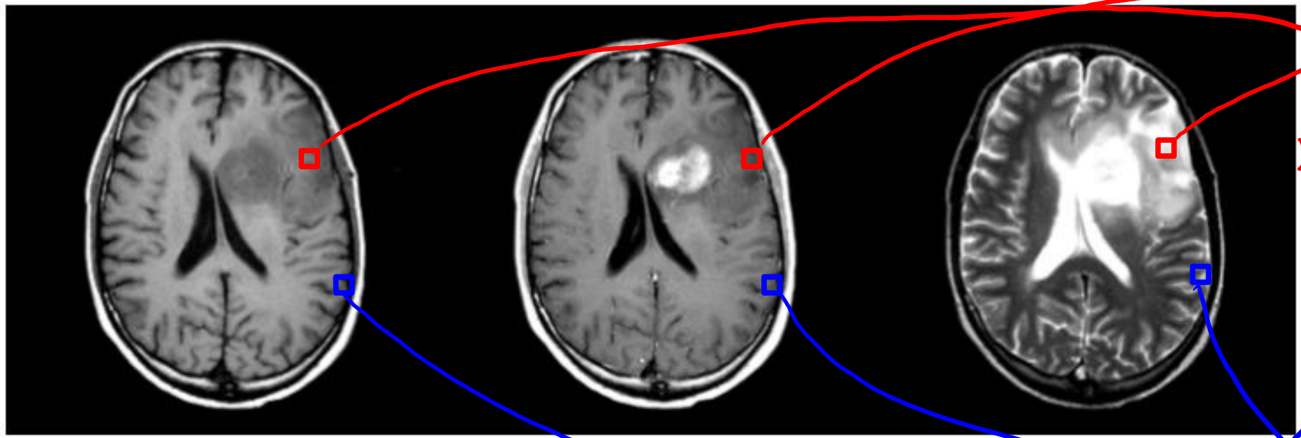


$$x_i = (98, 187, 246) \qquad y_i = \text{"tumour"}$$

- We can formulate predicting y_i given x_i as supervised learning.
- But it does not work at all with these features.

Need to Summarize Local Context

- The individual pixel intensity values are almost meaningless:
  - The same x_i could lead to different y_i.



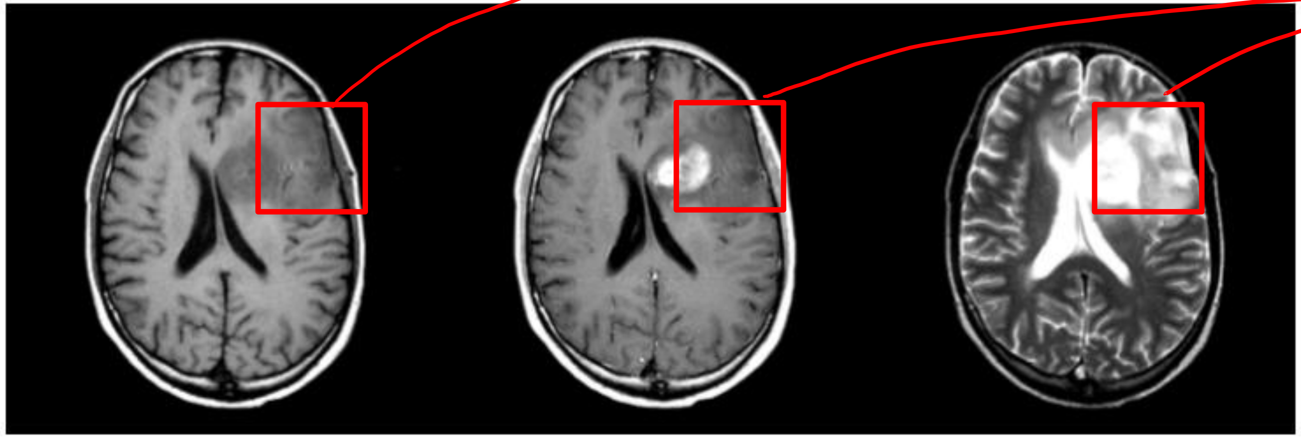
$x_i = (87, 54, 173)$ $y_i =$ "tumour"

$x_i = (87, 54, 173)$ $y_i =$ "normal"

- Intensities not standardized.
- Non-trivial overlap in signal for different tissue types.
- “Partial volume” effects at boundaries of tissue types.

Need to Summarize Local Context

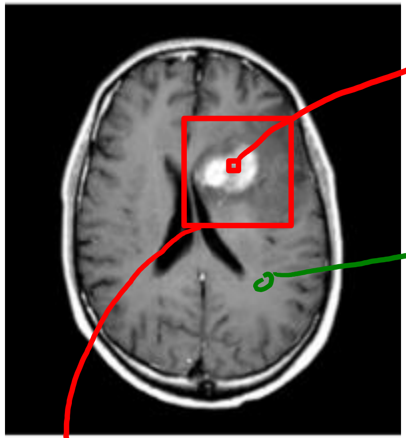
- We need to represent the "context" of the pixel (what is around it).



$x_i = (\overbrace{\ldots\ldots}, \overbrace{\ldots\ldots}, \overbrace{\ldots\ldots})$

  - Include all the values of neighbouring pixels as extra features?
    - Run into coupon collection problems: requires lots of data to find patterns.
  - Measure neighbourhood summary statistics (mean, variance, histogram)?
    - Variation on bag of words problem: loses spatial information present in voxels.
  - Standard approach uses convolutions to represent neighbourhood.

Example: Measuring "brightness" of an Area



- This pixel is in a "bright" area of the image, which reflects "bleeding" of tumour.
- But the actual numeric intensity value of the pixel is the same as in darker "gray matter" areas.

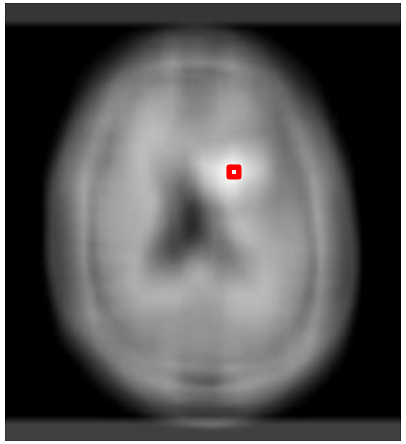
- I want a feature saying "this pixel is in a bright area of the image".
  - This will us help identify that it's a tumour pixel.

- Obvious way to measure brightness in area: take average pixel intensity in "neighbourhood".

$$z = \frac{1}{|nei|} \sum_{k \in nei} x_k$$

new feature is average value in neighbourhood.

- Applying this "averaging" to every pixel gives a new image:

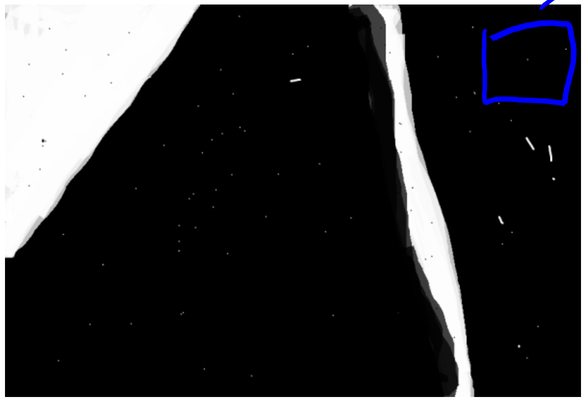


- We can use "pixel value in new image" as a new feature.
  - New feature helps identify if pixel is in a "bright" area.

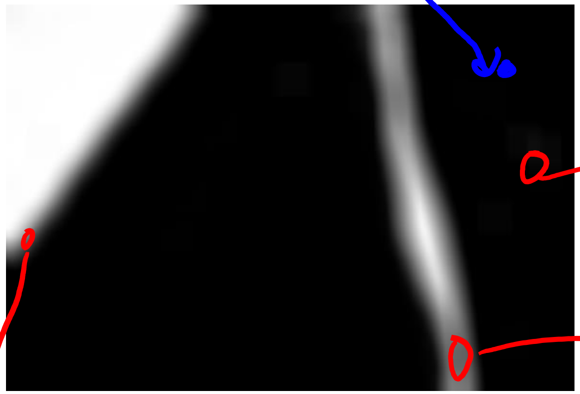
The annoying thing about squares

- “Take the average of a square window” loses spatial information.
- Example:

replace each pixel by the average value of "window"



Take average



Weird stuff

Pixels far away from "edge" become bright.

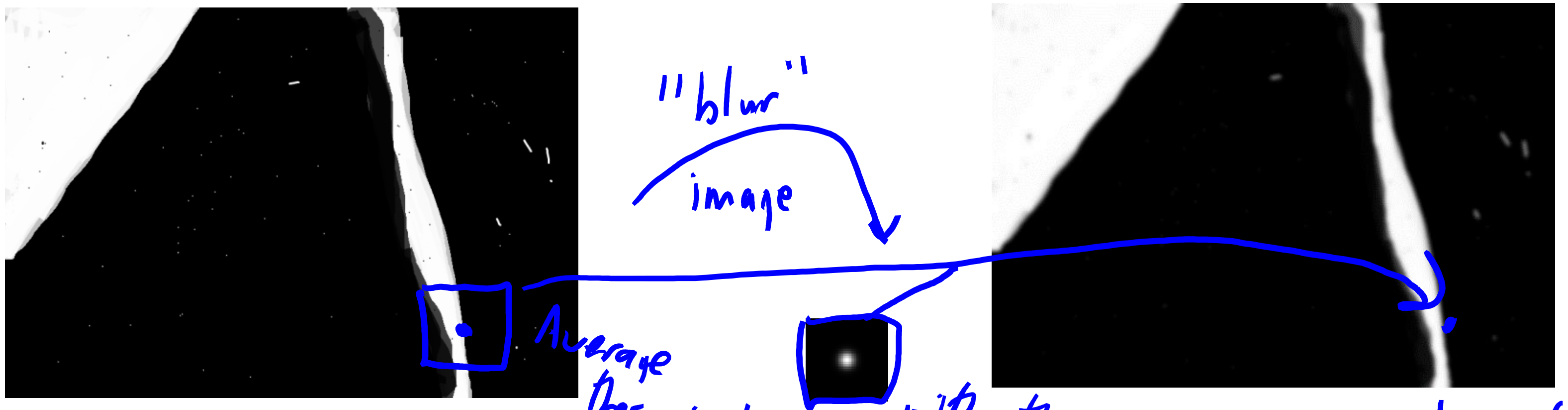
Noticeable features are "averaged out"

-average is higher, but location is lost.

Dark because line is "surrounded" by dark areas

Fixing the “square” issues

- Consider instead “blurring” the image.
  - Gets rid of “local” noise, but better preserves spatial information.



- How do you “blur”?
  - Take weighted average of window, putting more “weight” on “close” pixels:

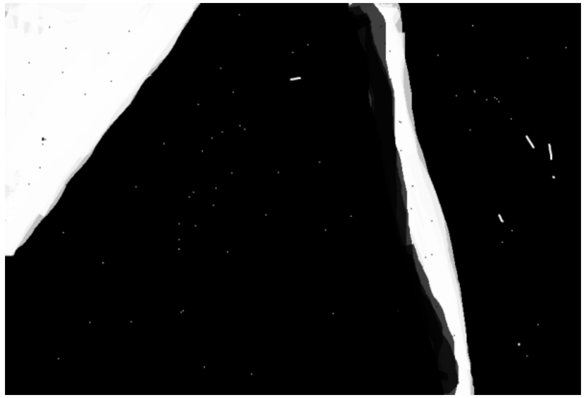
$$z = \sum w_k x_k$$

kernel ↳ weight on pixel k

(averaging is special case where all pixels get equal weight)

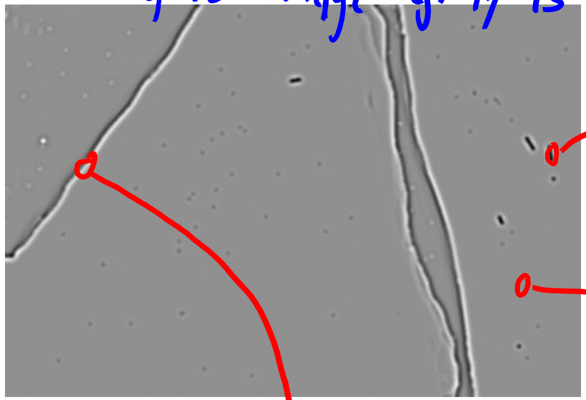
Fixing the “square” issues

- Another neat thing we can do: use negative weights.
  - These features can describe “differences” across space.



"Average" with positive and negative weights

"Signed" image: gray is 0.



"My neighbours are darker than me"

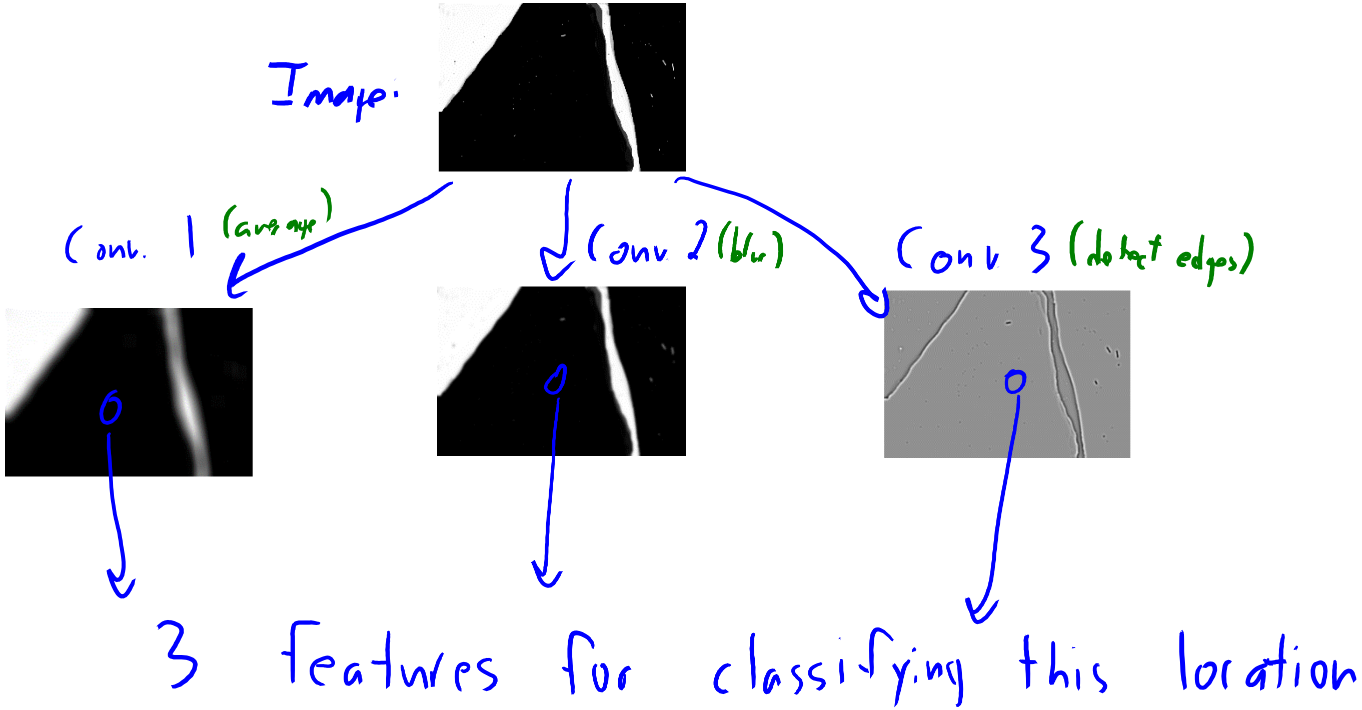
"My neighbours are the same as me"

Useful feature: "My neighbours are brighter than me"

- These “weighted averages of neighbours” are called “convolutions”.
  - I think of convolutions as the “words” that make up image regions.

Convolutions: Big Picture

- How do you use convolutions to get features?
  - Apply several different convolutions to your image.
  - Each convolution gives a different “image” value at each location.
  - Use theses different image values to give features at each location.

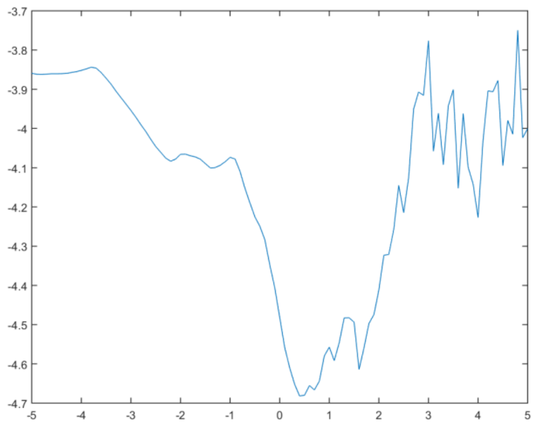


Convolutions: Big Picture

- What can features coming from convolutions represent?
  - Some filters give you an average value of the neighbourhood.
  - Some filters approximate the “first derivative” in the neighbourhood.
    - “Is there a change from low to dark to bright?”
    - “If so, from which direction in space?”
  - Some filters approximate the “second derivative” in the neighbourhood.
    - “Is there a spike or is the change speeding up?”
- Hope: we can characterize “what happens in a neighbourhood”, with just a few numbers.

1D Convolution Example

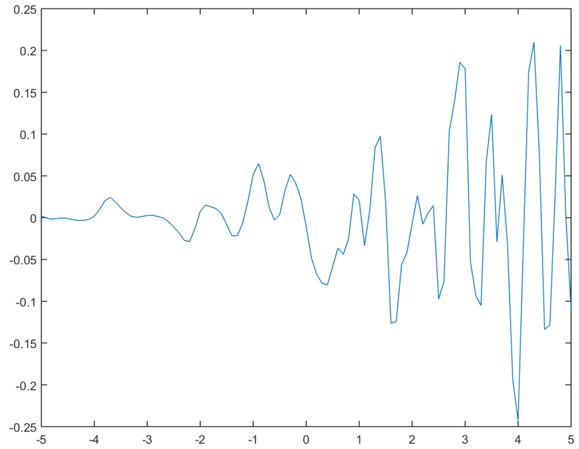
- Consider a 1D "signal" (maybe from sound):
  - We will come back to images later.



- For each "time":
  - Compute dot-product of signal at surrounding times with a "filter" of weights.

$w = [-0.1416 \;\; -0.1781 \;\; -0.2746 \;\; 0.1640 \;\; 0.8607 \;\; 0.1640 \;\; -0.2746 \;\; -0.1781 \;\; -0.1416]$

- This gives a new "signal":
  - Measures a property of "neighbourhood".
  - This particular filter shows a local "how spiky " value.



1D Convolution (notation is specific to this lecture)

- 1D convolution input:
  - Signal 'x' which is a vector length 'n'.
    - Indexed by i=1,2,…,n.
  - Filter 'w' which is a vector of length '2m+1':
    - Indexed by i=-m,-m+1,…-2,0,1,2,…,m-1,m

$x = [0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13]$

$$w = \begin{array}{ccccc} [0 & -1 & 2 & -1 & 0] \\ w_{-2} & w_{-1} & w_0 & w_1 & w_2 \end{array}$$

- Output is a vector of length 'n' with elements:

$$z_i = \sum_{j=-m}^{m} w_j x_{i+j}$$

  - You can think of this as centering w at position 'i', and taking a dot product of 'w' with that "part" x_i.

1D Convolution

- 1D convolution example:
  - Signal ‘x’:

0	**1**	**1**	**2**	**3**	**5**	**8**	**13**

  - Filter ‘w’:

0	**-1**	**2**	**-1**	**0**

  - Convolution ‘z’:

1D Convolution

- 1D convolution example:
  - Signal 'x':

0	1	1	2	3	5	8	13

  - Filter 'w':

0	-1	2	-1	0

  - Convolution 'z':

		-1					

take dot-product $(0 \cdot 0 + 1 \cdot (-1) + 1 \cdot 2 + 2 \cdot (-1) + 3 \cdot 0)$

1D Convolution

- 1D convolution example:
  - Signal 'x':

0	1	1	2	3	5	8	13

  - Filter 'w':

0	-1	2	-1	0

  - Convolution 'z':

		-1	0				

1D Convolution

- 1D convolution example:
  - Signal ‘x’:

0	1	1	2	3	5	8	13

  - Filter ‘w’:

0	-1	2	-1	0

  - Convolution ‘z’:

		-1	0	-1			

1D Convolution

- 1D convolution example:
  - Signal ‘x’:

0	1	1	2	3	5	8	13

  - Filter ‘w’:

0	-1	2	-1	0

  - Convolution ‘z’:

		-1	0	-1	-1		

1D Convolution Examples

- Examples:
  - "Identity"

    $\hookrightarrow w = [0 \quad 1 \quad 0]$

  - "Translation"

    $\hookrightarrow w = [0 \quad 0 \quad 1]$

Let $x = [0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13]$

$z = [0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13]$

$0 \cdot x_0 + 1 \cdot x_1 + 0 \cdot x_2$ $\quad 0 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3$

$z = [1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad ?]$

$0 \cdot x_0 + 0 \cdot x_1 + 1 \cdot x_2$

1D Convolution Examples

- Examples:
  - "Identity"

    $w = [0 \quad 1 \quad 0]$

  - "Local Average"

    $w = [\tfrac{1}{3} \quad \tfrac{1}{3} \quad \tfrac{1}{3}]$

Let $x = [0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13]$

average

average

$z = [? \quad \tfrac{2}{3} \quad 1\tfrac{1}{3} \quad 2 \quad 3\tfrac{1}{3} \quad 5\tfrac{1}{3} \quad 8\tfrac{2}{3} \quad ?]$

Boundary Issue

- What can we do about the "?" at the edges?

If $x = [0\ 1\ 1\ 2\ 3\ 5\ 8\ 13]$ and $w = [\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}]$ then $z = [?\ \frac{2}{3}\ 1\frac{1}{3}\ 2\ 3\frac{1}{3}\ 5\frac{1}{3}\ 8\frac{2}{3}\ ?]$

- Can assign values past the boundaries:
  - "Zero": $x = 0\ 0\ 0\ [0\ 1\ 1\ 2\ 3\ 5\ 8\ 13]\ 0\ 0\ 0$
  - "Replicate": $x = 0\ 0\ 0\ [0\ 1\ 1\ 2\ 3\ 5\ 8\ 13]\ 13\ 13\ 13$
  - "Mirror": $x = 2\ 1\ 1\ [0\ 1\ 1\ 2\ 3\ 5\ 8\ 13]\ 8\ 5\ 3$
- Or just ignore the "?" values and return a shorter vector:

$$z = [\tfrac{2}{3}\ 1\tfrac{1}{3}\ 2\ 3\tfrac{1}{3}\ 5\tfrac{1}{3}\ 8\tfrac{2}{3}]$$

Formal Convolution Definition

- We've defined the convolution as:

$$z_i = \sum_{j=-m}^{m} w_j x_{i+j}$$

- In other classes you may see it defined as:

$$z_i = \sum_{j=-m}^{m} w_j x_{i-j}$$

(reverses 'w')

$$z_i = \int_{-\infty}^{\infty} w_j x_{i-j} \, dj$$

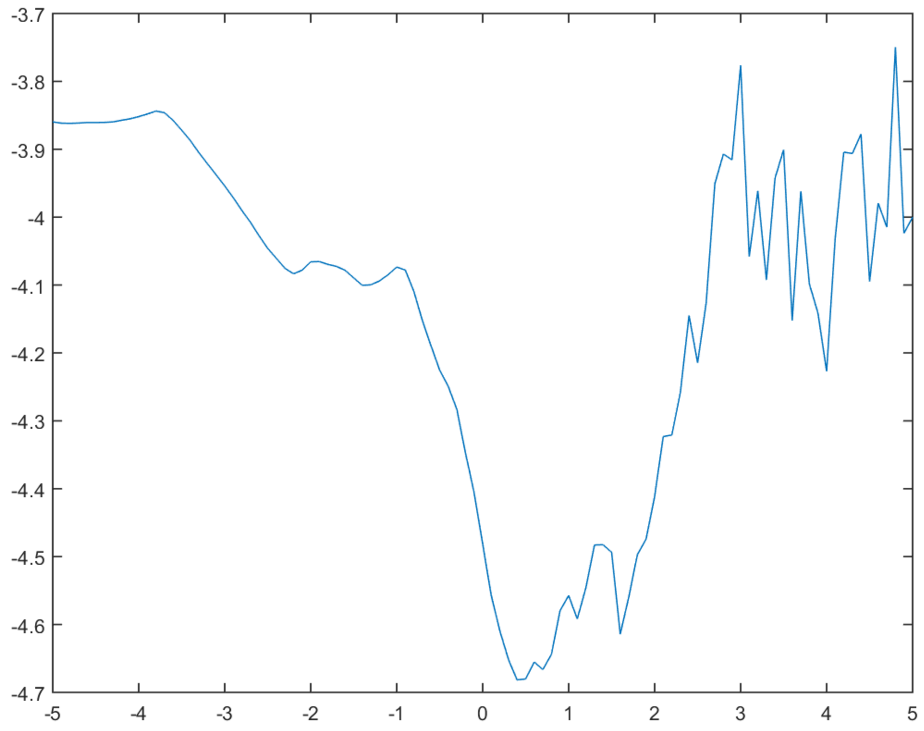
(assumes signal + filter are continuous)

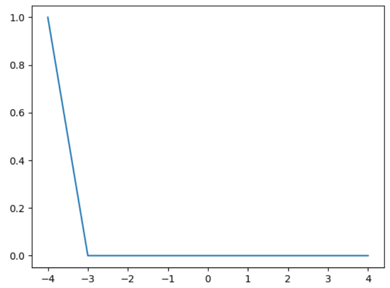
- For simplicity we use "+" instead of "-",
  and assume 'w' and 'x' are sampled at discrete points (not functions).
- But keep this mind if you read about convolutions elsewhere.

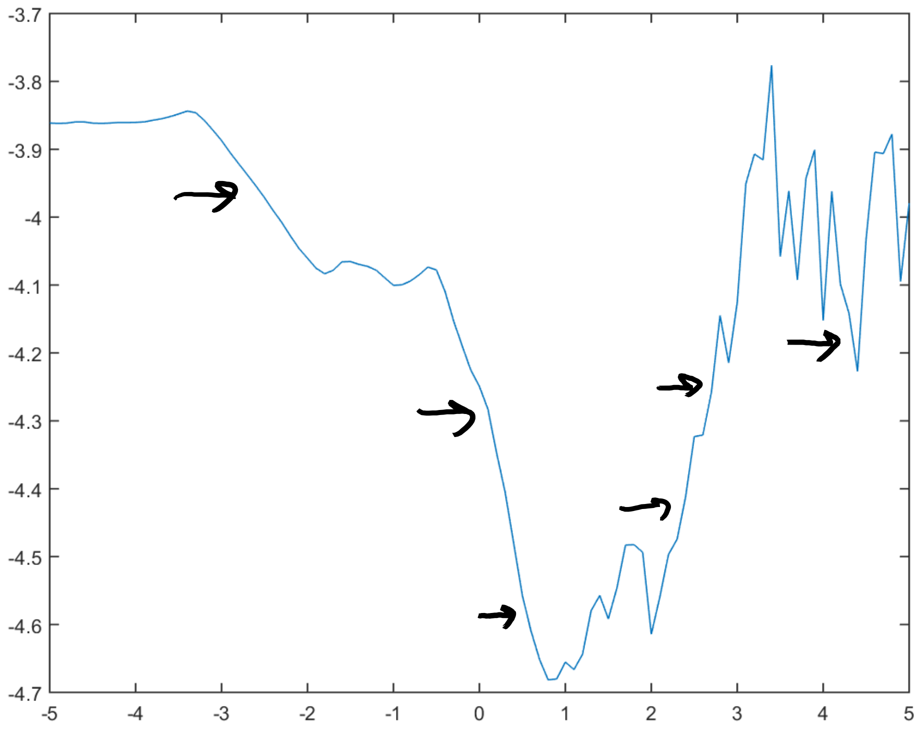
1D Convolution Examples

- Translation convolution shift signal:
  - “What is my neighbour’s value?”

$w = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$



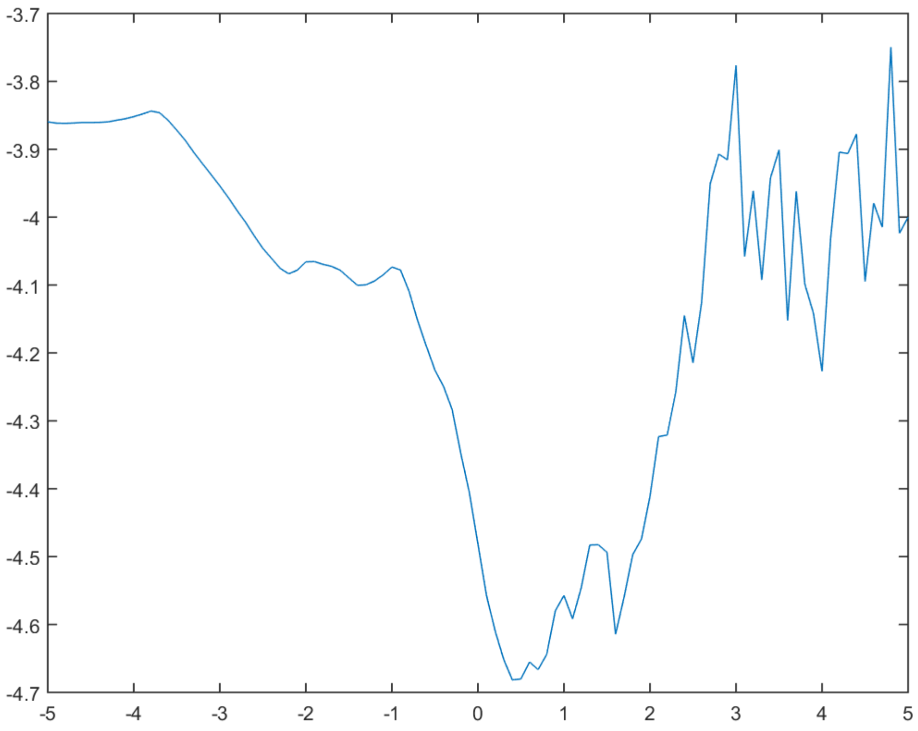


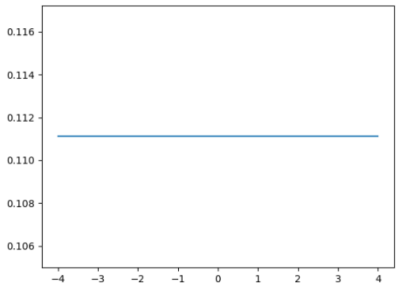


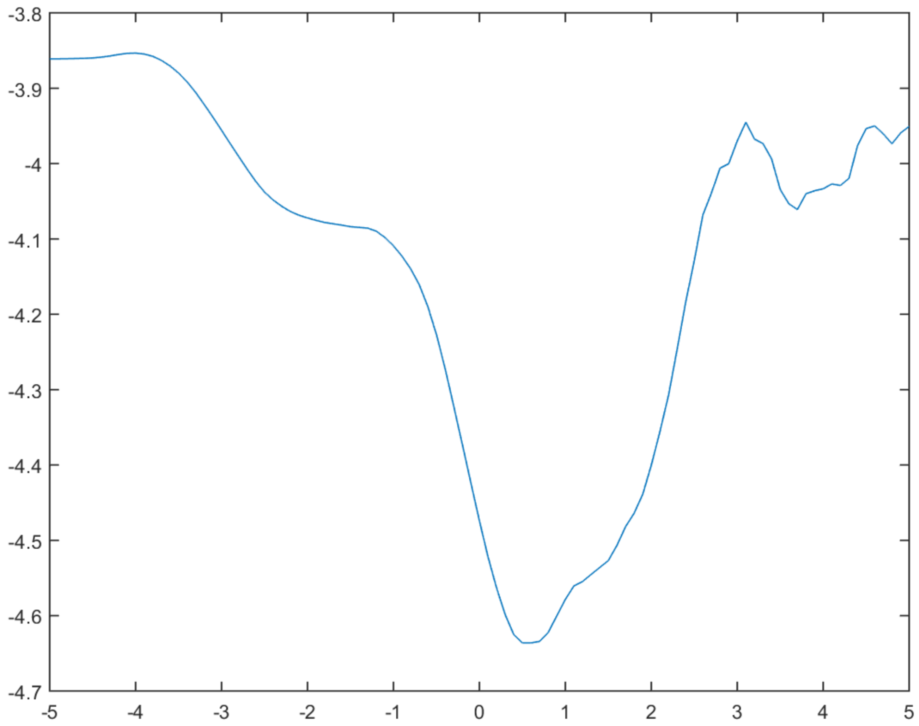
1D Convolution Examples

- Averaging convolution ("is signal generally high in this region?")
  - Less sensitive to noise (or spikes) than raw signal.

$w = [\,1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9 \;\; 1/9\,]$





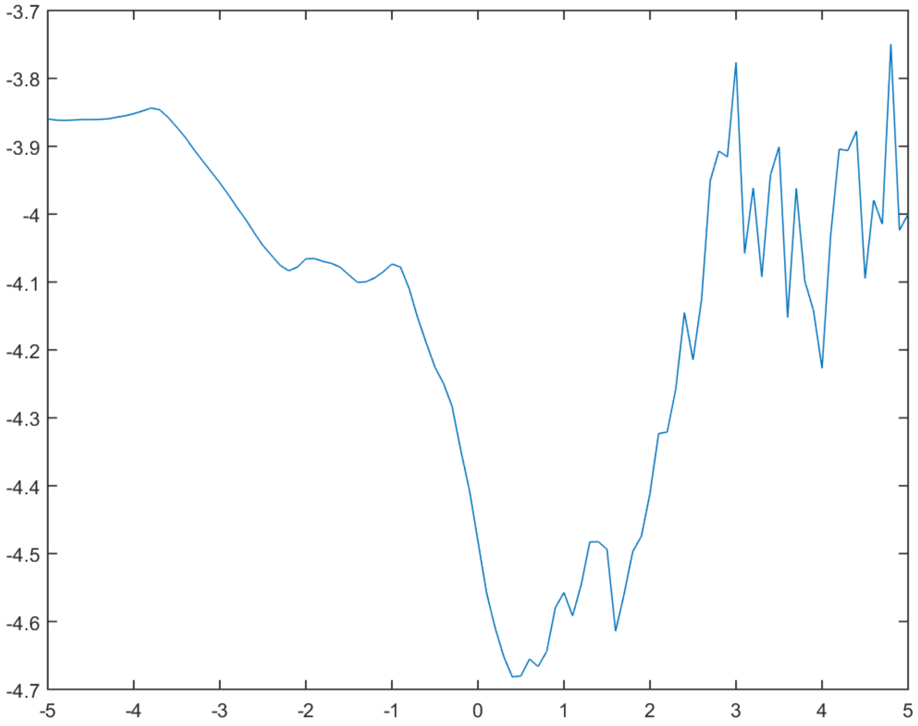


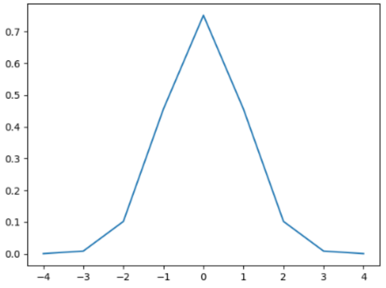
1D Convolution Examples

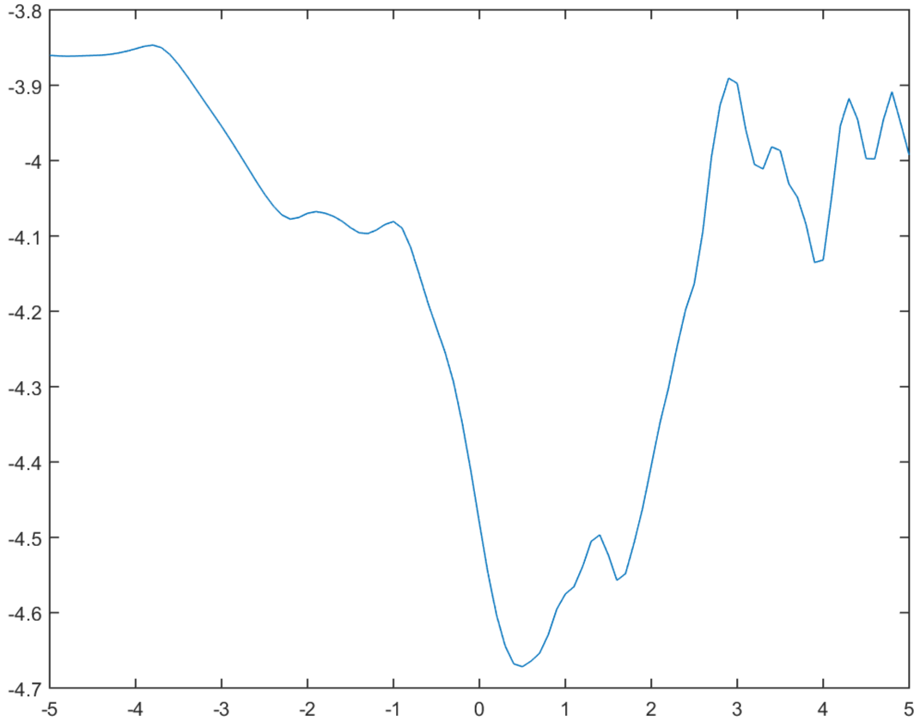
- Gaussian convolution ("blurring"): $w_i \propto \exp\left(-\frac{i^2}{2\sigma^2}\right)$
  - Compared to averaging it's more smooth and maintains peaks better.

$W = [0.0001 \quad 0.0044 \quad 0.0540 \quad 0.2420 \quad 0.3989 \quad 0.2420 \quad 0.0540 \quad 0.0044 \quad 0.0001]$

$(\sigma = 1, \; m = 4)$



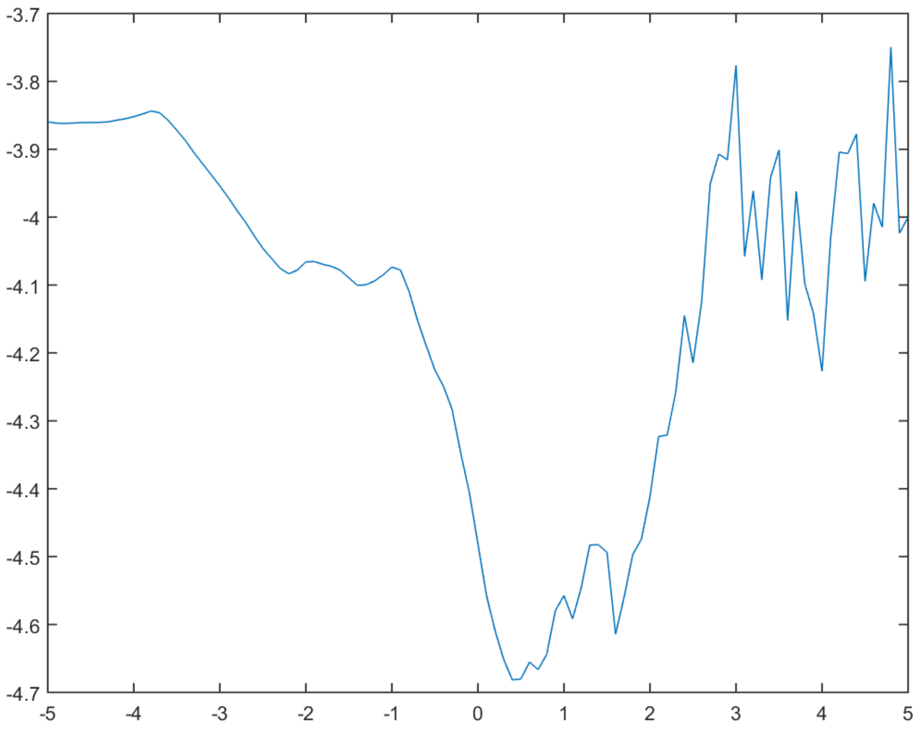


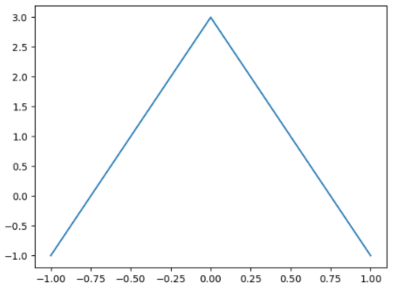


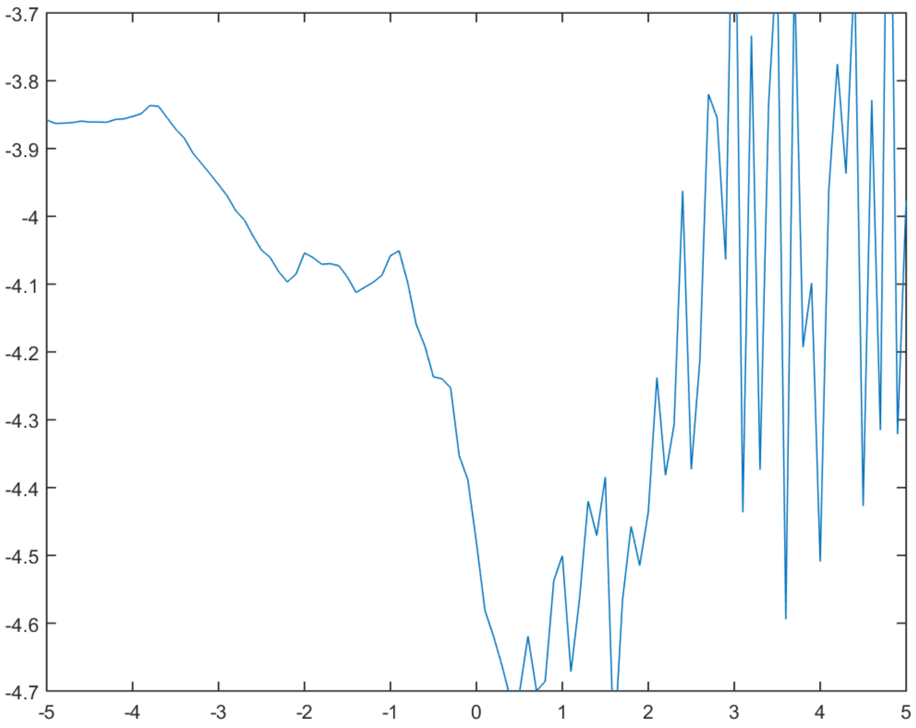
1D Convolution Examples

- Sharpen convolution enhances peaks.
  - An "average" that places negative weights on the surrounding pixels.

$w = [-1 \quad 3 \quad -1]$



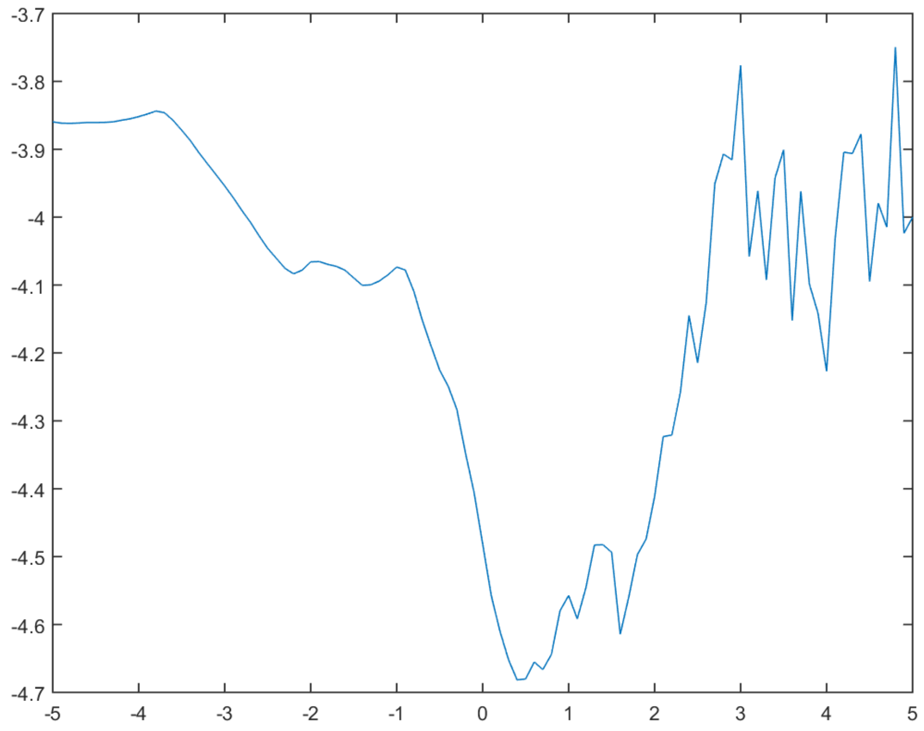


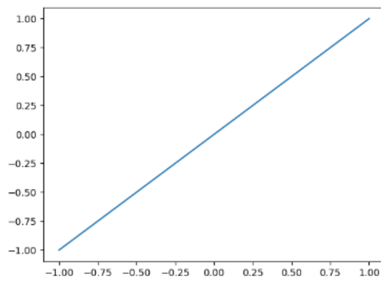


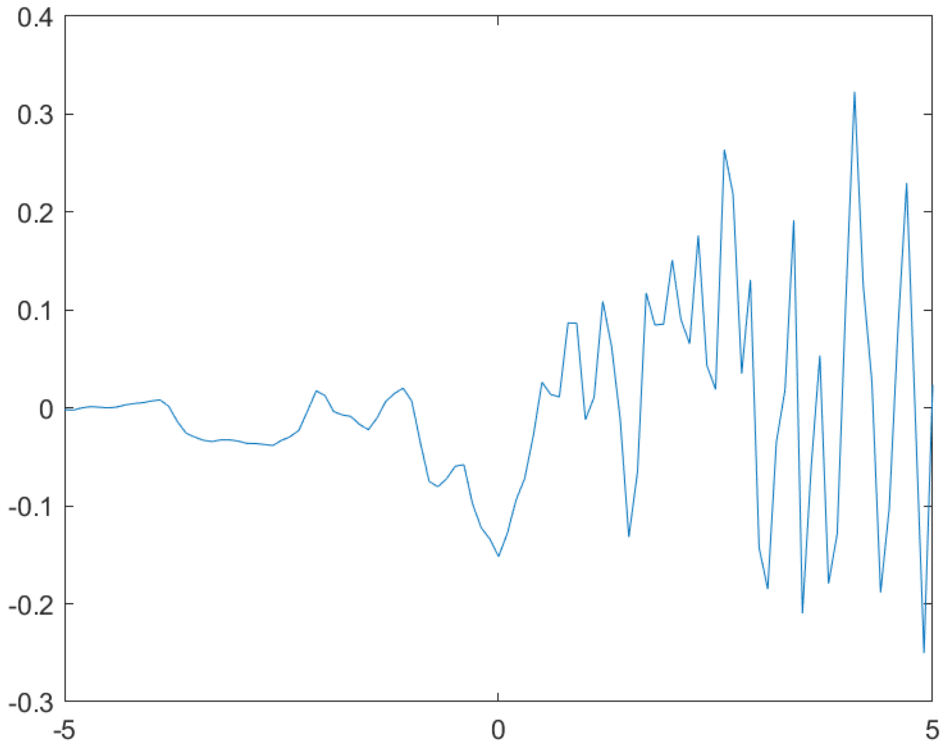
1D Convolution Examples

- Centered difference convolution approximates first derivative:
  - Positive means change from low to high (negative means high to low).

$w = [-1 \quad 0 \quad 1]$

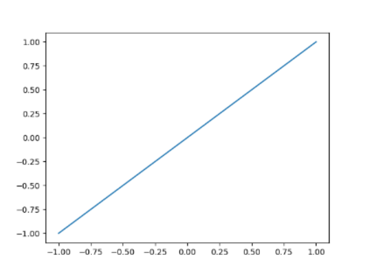




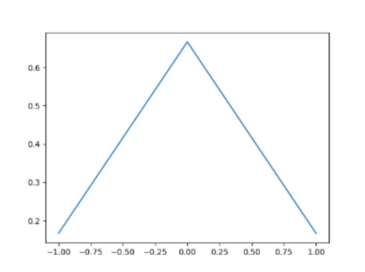


Digression: Derivatives and Integrals

- Numerical derivative approximations can be viewed as filters:
  - Centered difference: [-1, 0, 1] (derivativeCheck in findMin).



- Numerical integration approximations can be viewed as filters:
  - "Simpson's" rule: [1/6, 4/6, 1/6] (a bit like Gaussian filter).

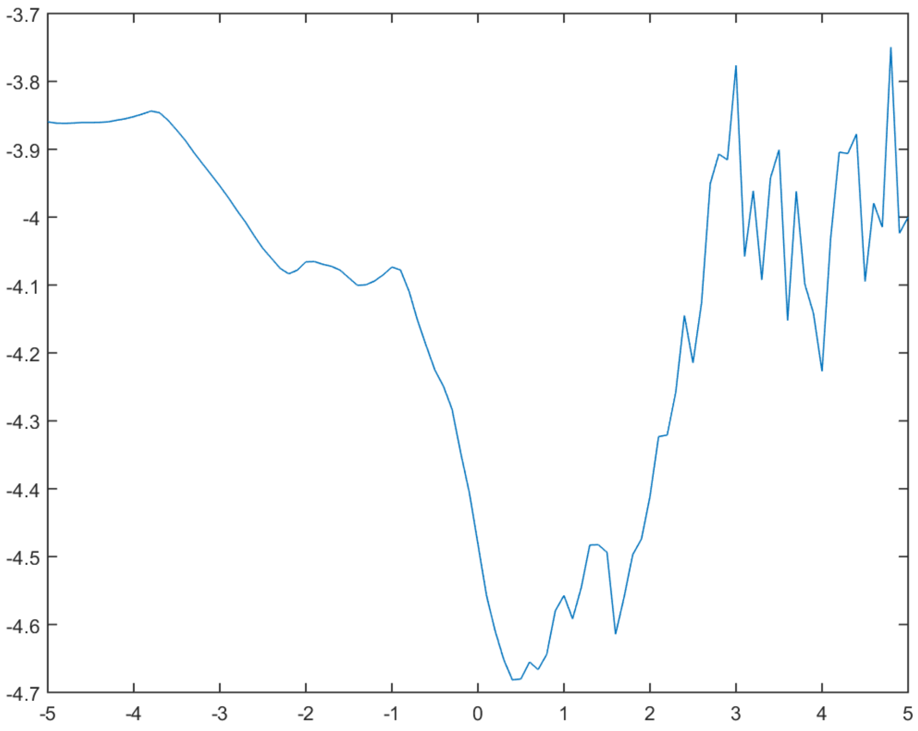


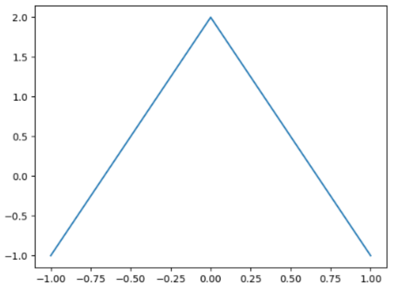
- Derivative filters add to 0, integration filters add to 1,
  - For constant function, derivative should be 0 and average = constant.

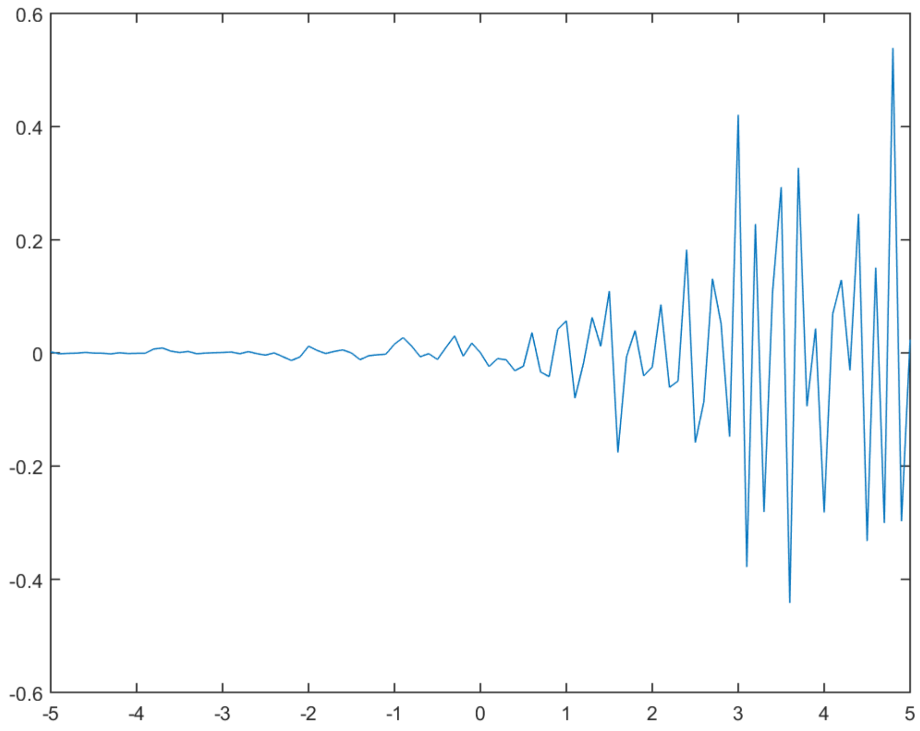
1D Convolution Examples

- Laplacian convolution approximates second derivative:
  - "Sum to zero" filters "respond" if input vector looks like the filter

$w = [-1 \quad 2 \quad -1]$







Laplacian of Gaussian Filter

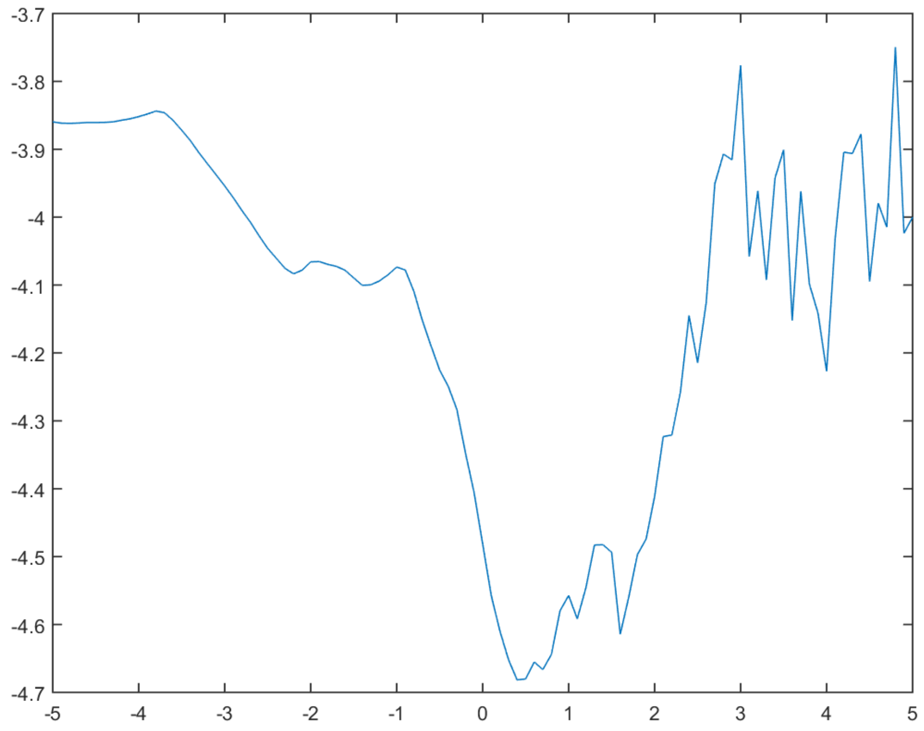
- Laplacian of Gaussian is a smoothed 2nd-derivative approximation:

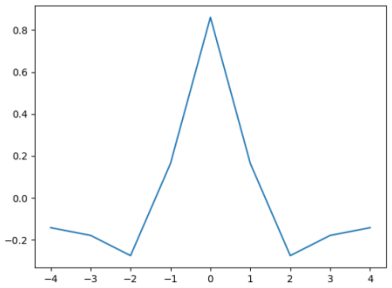
$$w_i = \left(1 - \frac{i^2}{2\sigma^2}\right)\exp\left(-\frac{i^2}{2\sigma^2}\right)$$

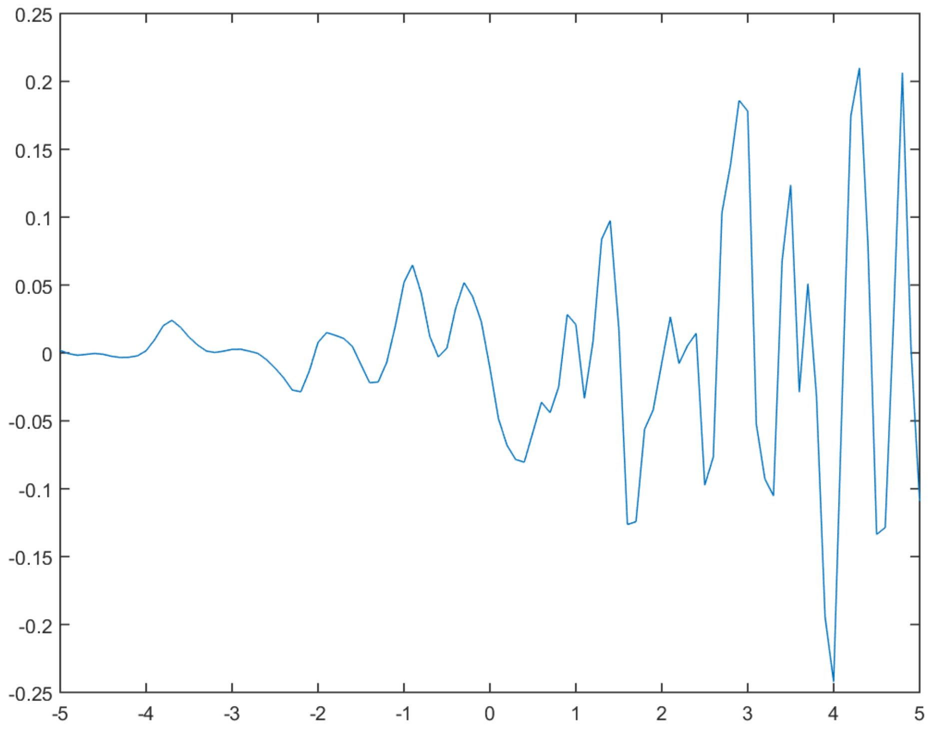
(then subtract mean)

$w = [-0.1416\ \ -0.1781\ \ -0.2746\ \ 0.1640\ \ 0.8607\ \ 0.1640\ \ -0.2746\ \ -0.1781\ \ -0.1416]$

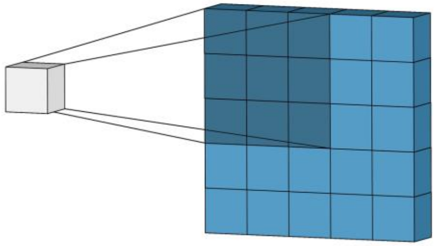
$(\sigma^2 = 1,\ m = 4)$







Images and Higher-Order Convolution

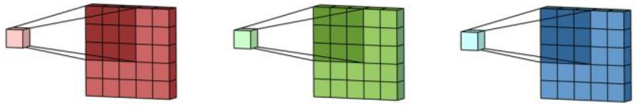


- 2D convolution:
  - Signal ‘x’ is the pixel intensities in an ‘n’ by ‘n’ image.
  - Filter ‘w’ is the pixel intensities in a ‘2m+1’ by ‘2m+1’ imag
- The 2D convolution is given by:

$$z[i_1, i_2] = \sum_{j_1=-m}^{m} \sum_{j_2=-m}^{m} w[j_1, j_2]\, x[i_1 + j_1, i_2 + j_2]$$

- 3D and higher-order convolutions are defined similarly.

$$z[i_1, i_2, i_3] = \sum_{j_1=-m}^{m} \sum_{j_2=-m}^{m} \sum_{j_3=-m}^{m} w[j_1, j_2, j_3]\, x[i_1 + j_1, i_2 + j_2, i_3 + j_3]$$



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

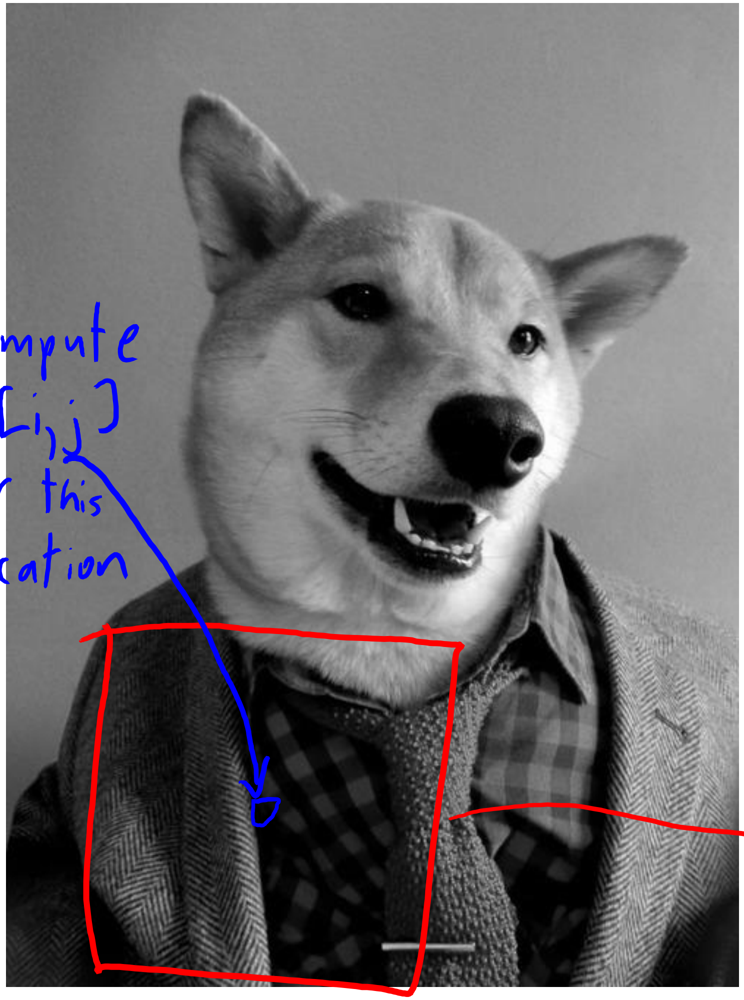
Image Convolution Examples

x

Identity convolution:

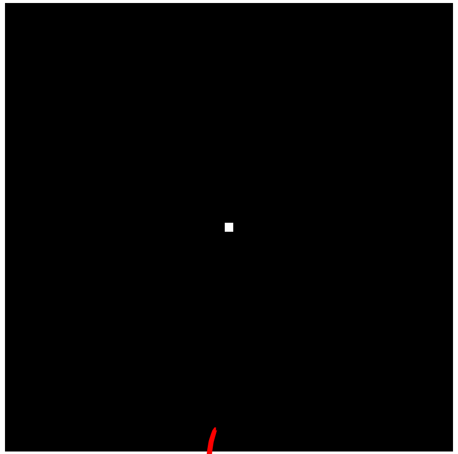
(zeroes with a '1' at $w_{0,0}$)

Compute z[i,j] for this location



*

w



=

z

multiply element-wise and add up result to get

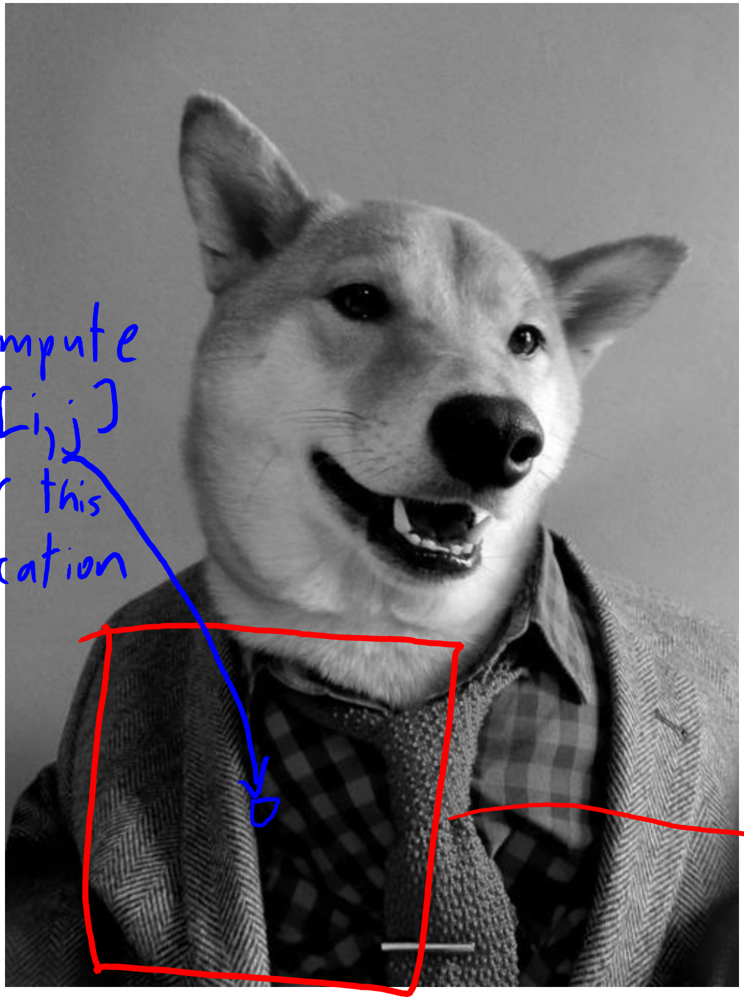
z[i,j]

Image Convolution Examples

x

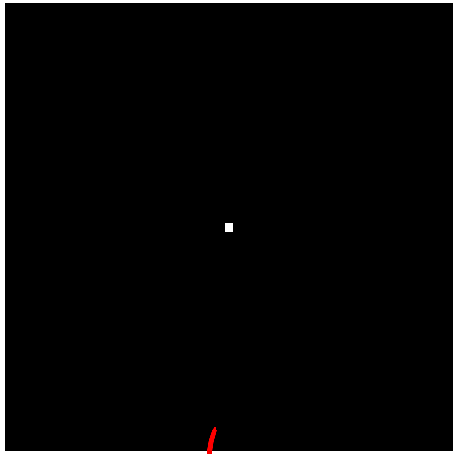
Identity convolution:
(zeroes with a '1' at $w_{0,0}$)

Compute
$z[i,j]$
for this
location



$*$

w



$=$

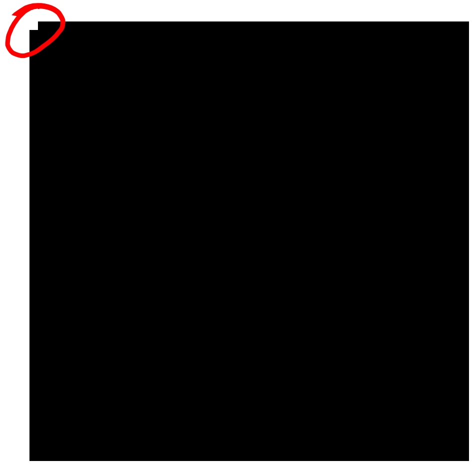
z

multiply element-wise
and add up result to get

Image Convolution Examples

Translation Convolution:

$*$



Boundary: "zero"

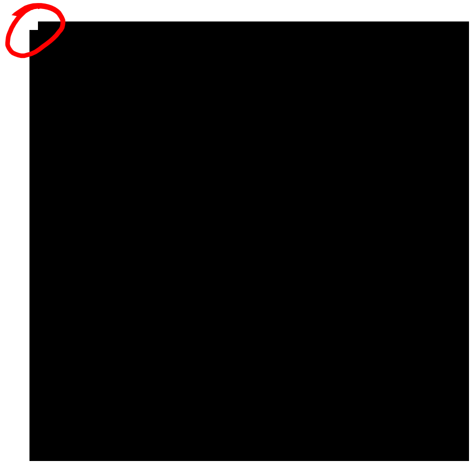
$=$

0

Image Convolution Examples

Translation Convolution:

*



=

repeats

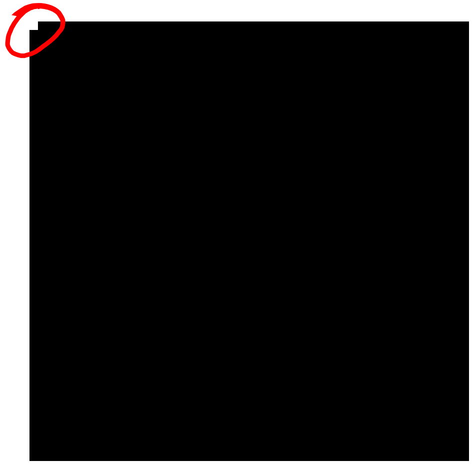
repeats

Boundary: "replicate"

Image Convolution Examples

Translation Convolution:

*



=

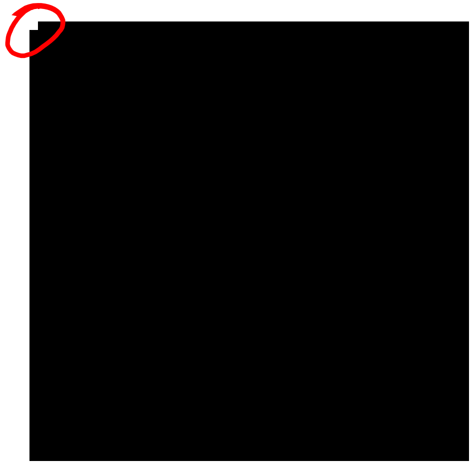
Boundary: "mirror"

flips

Image Convolution Examples

Translation Convolution:

$*$



$=$

Boundary: "ignore"

Summary

- Text features (beyond bag of words): trigrams, lexical, stem, shape.
  - Try to capture important invariances in text data.
- Global vs. local features allow “personalized” predictions.
- Convolutions are flexible class of signal/image transformations.
  - Can approximate directional derivatives and integrals at different scales.
  - Max(convolutions) can yield features invariant to some transformations.

- Next time:
  - A trick that lets you find gold and use the polynomial basis with $d > 1$.

Cyclic Features

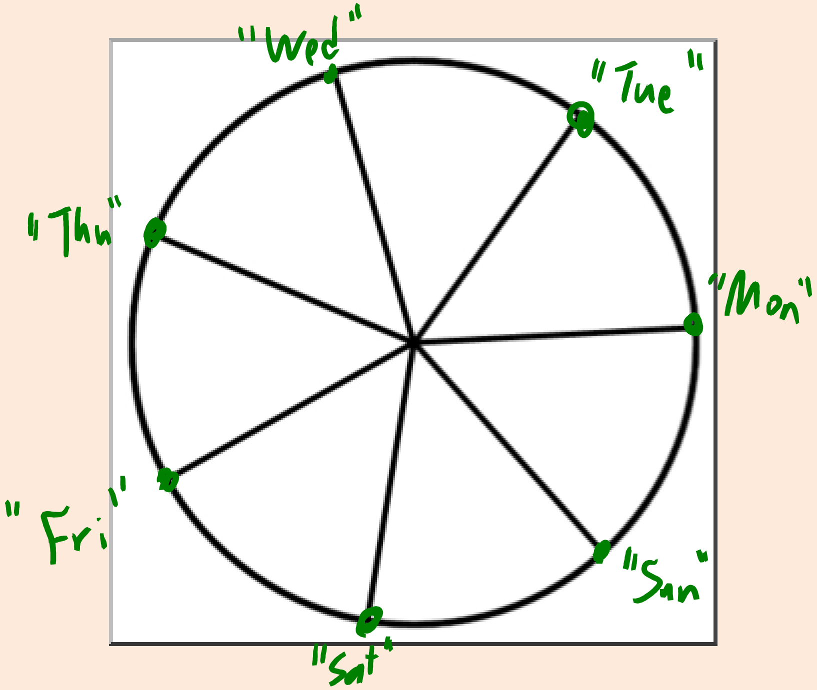
- Cyclic features arise in many settings, especially with times:

Time	Day	Date	Month	Year
12:05pm	Wed	29	Jul	15
10:20am	Sun	24	Apr	16
9:10am	Tue	3	May	16
11:20am	Sun	15	Jun	18
10:15pm	Thu	8	Aug	19

- Could use ordinal: “Jan”->1, “Feb”->2, “Mar”->3, and so on.
  - Reflects ordering of months
  - But this says that “Jan” and “Dec” are far.
  - We might want to incorporate the “cycle” that “1” comes after “12”.

Cyclic Features

- One way to model cyclic features is as coordinates on unit circle.
  - Dividing circumference evenly across the cyclic values.



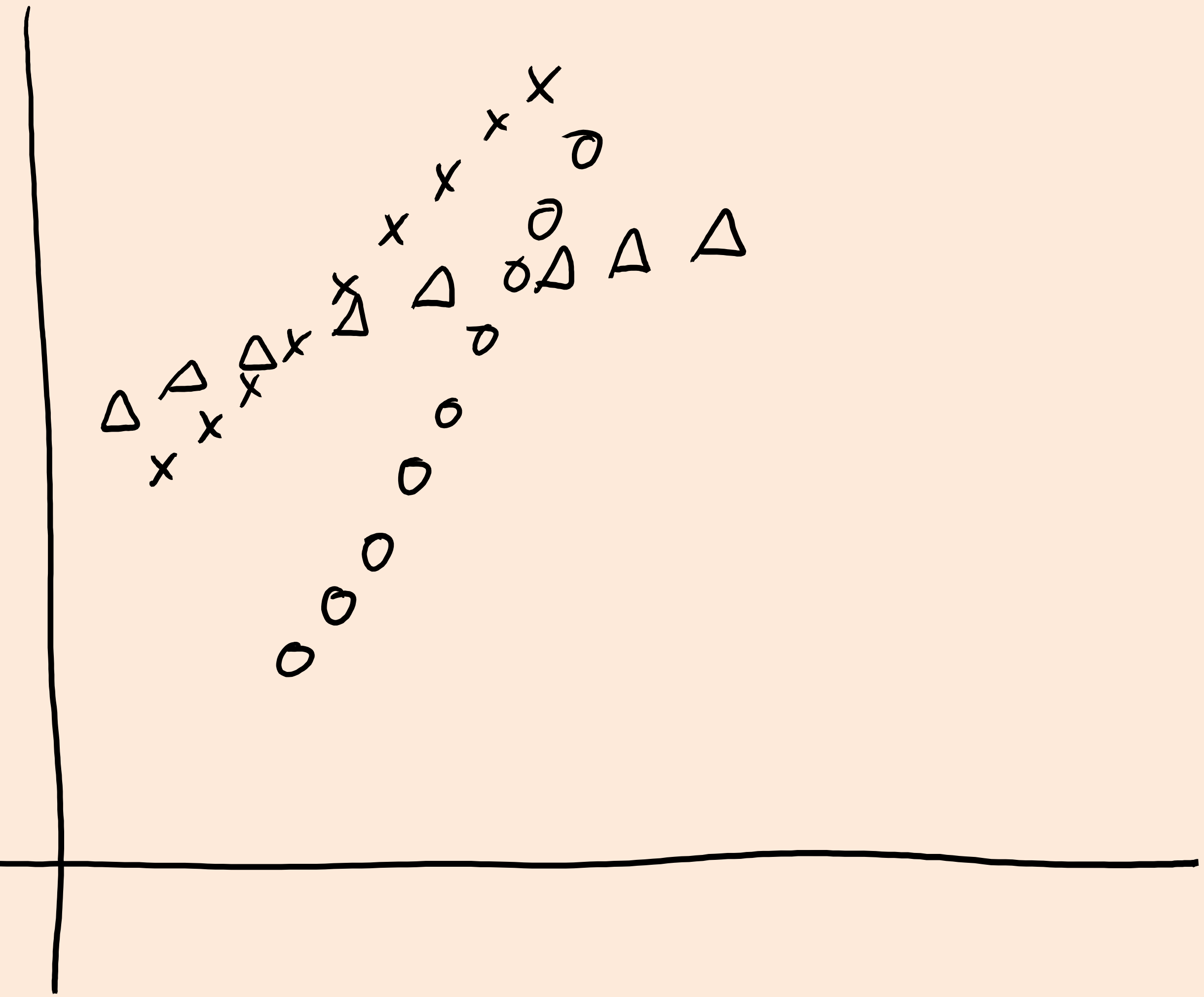
- Replace "Day" with the x-coordinate and y-coordinate (2 features).
  - Reflects that "Mon" is same distance from "Tue" as it is from "Sun".

https://www.abcteach.com/documents/clip-art-circle07-77-bw-i-abcteachcom-17022

Linear Models with Binary Features

$X =$

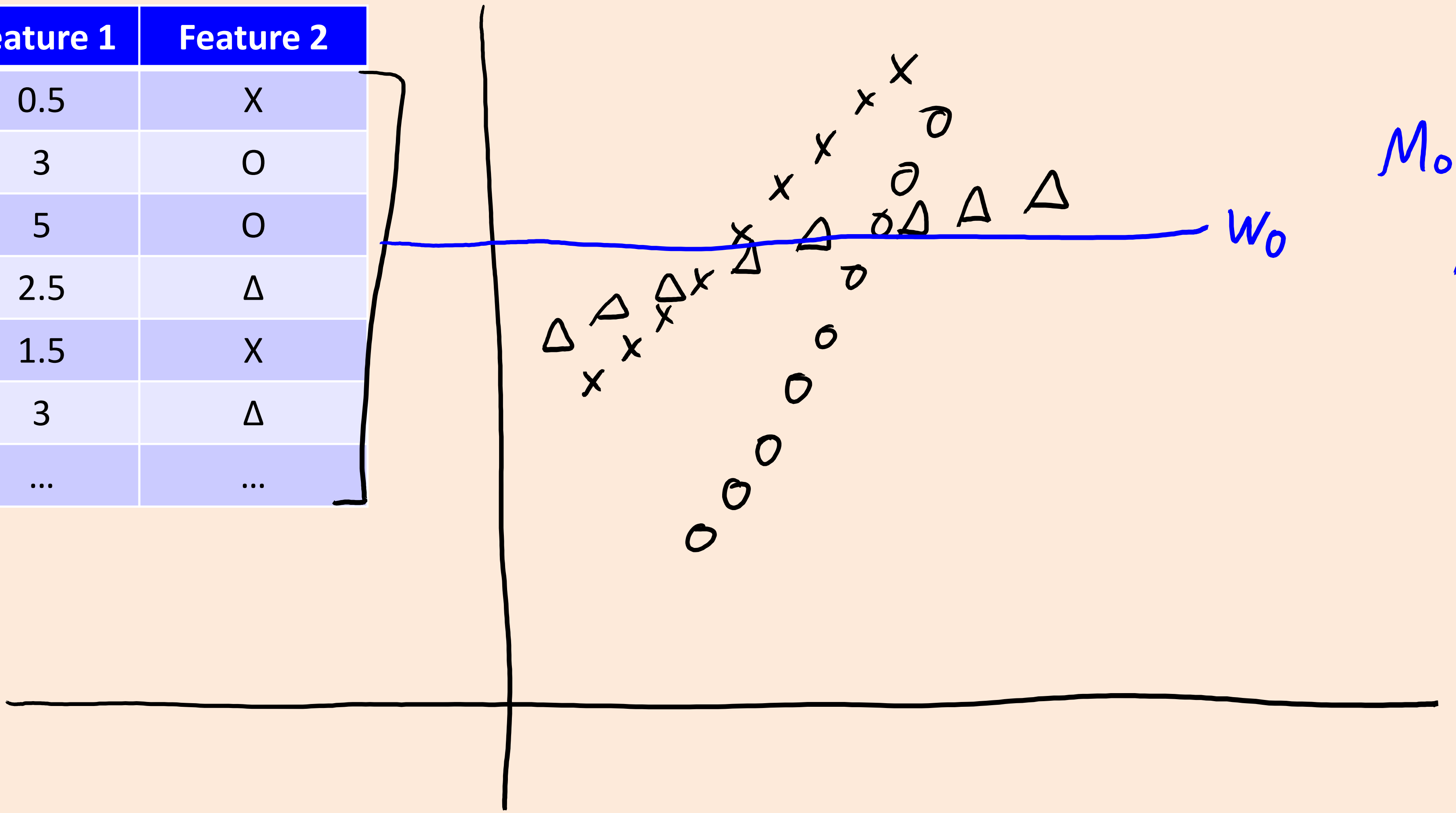
Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Linear Models with Binary Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



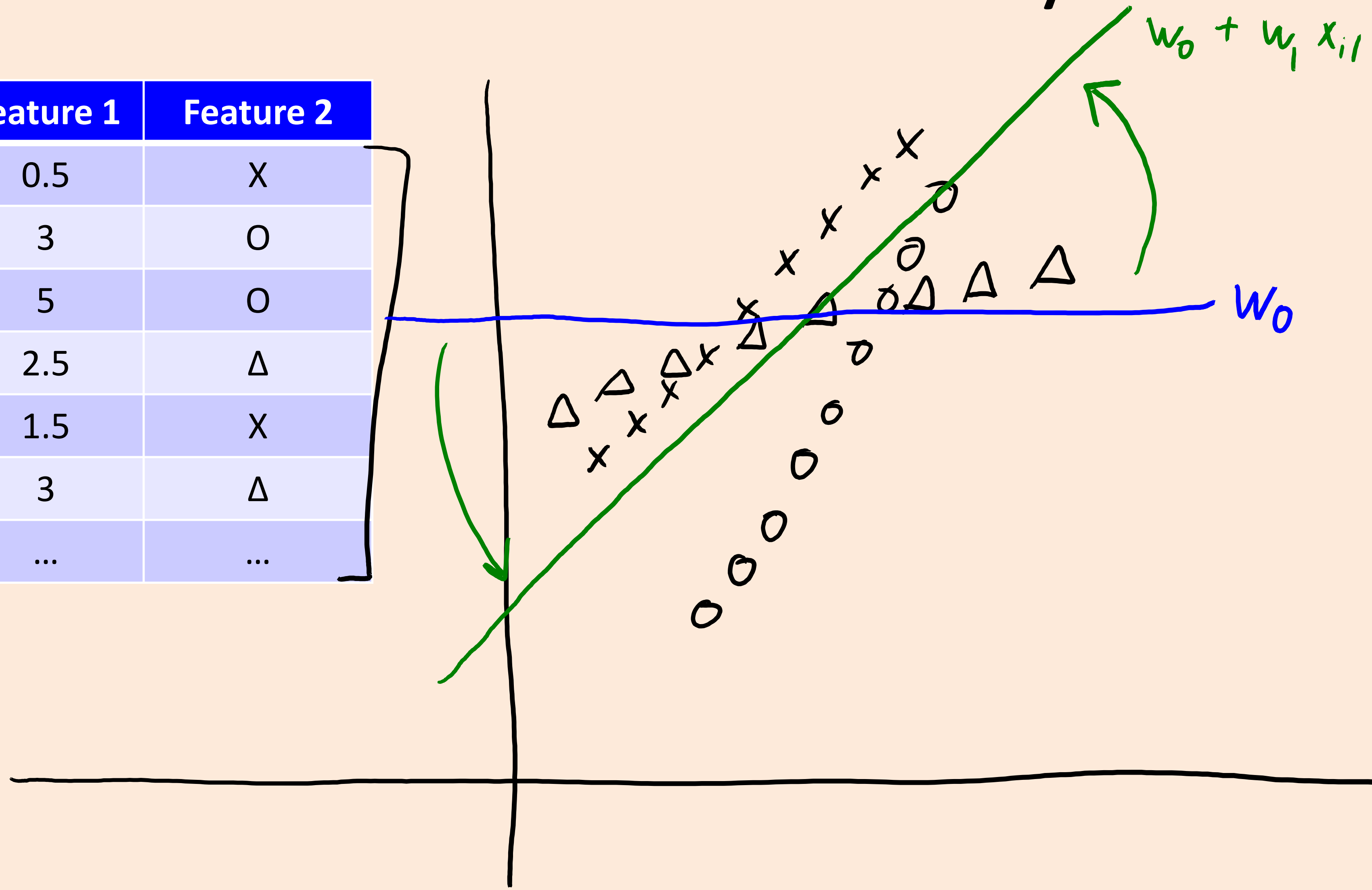
Model 1: only bias

$y_i = w_0$

Linear Models with Binary Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

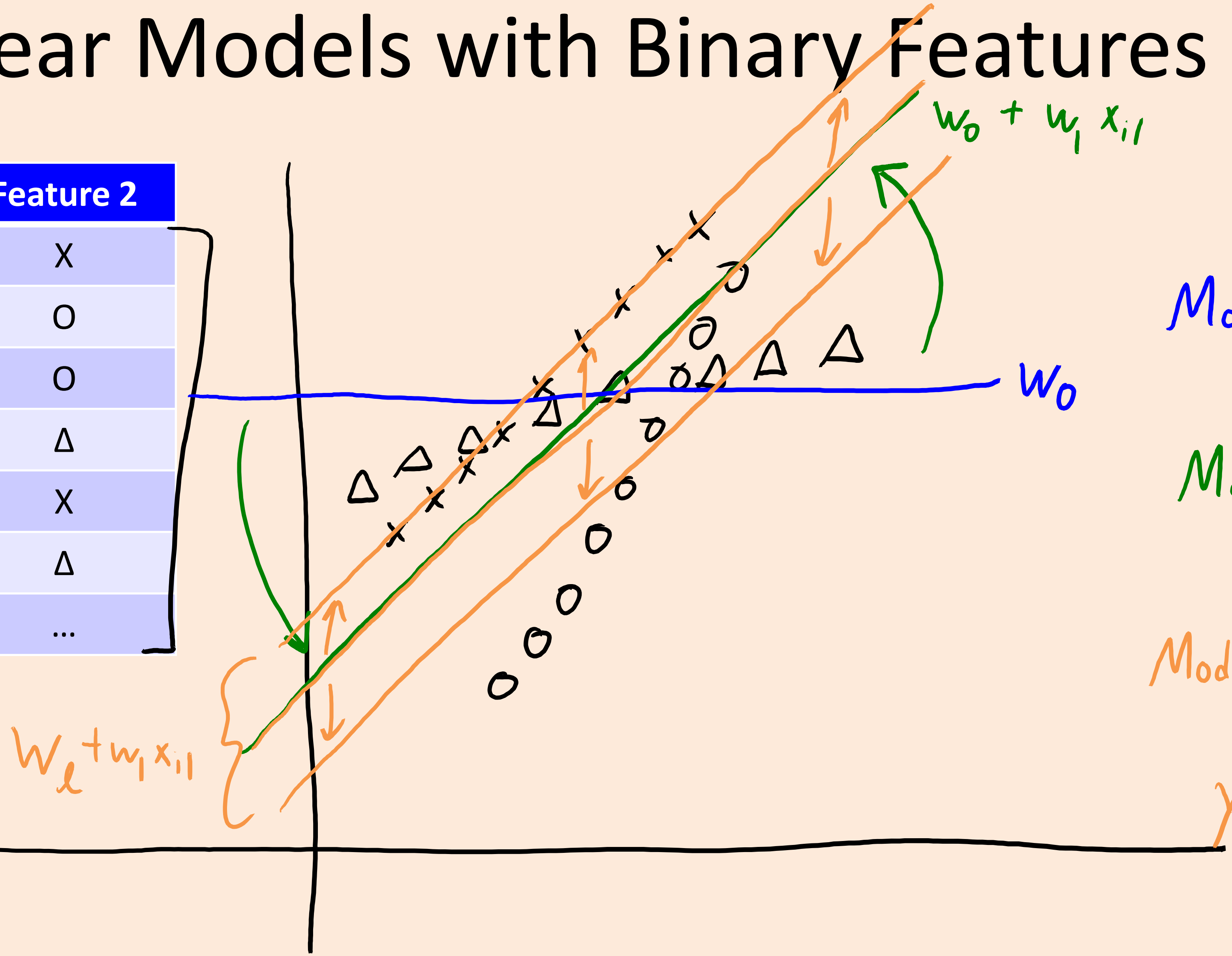
Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Linear Models with Binary Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

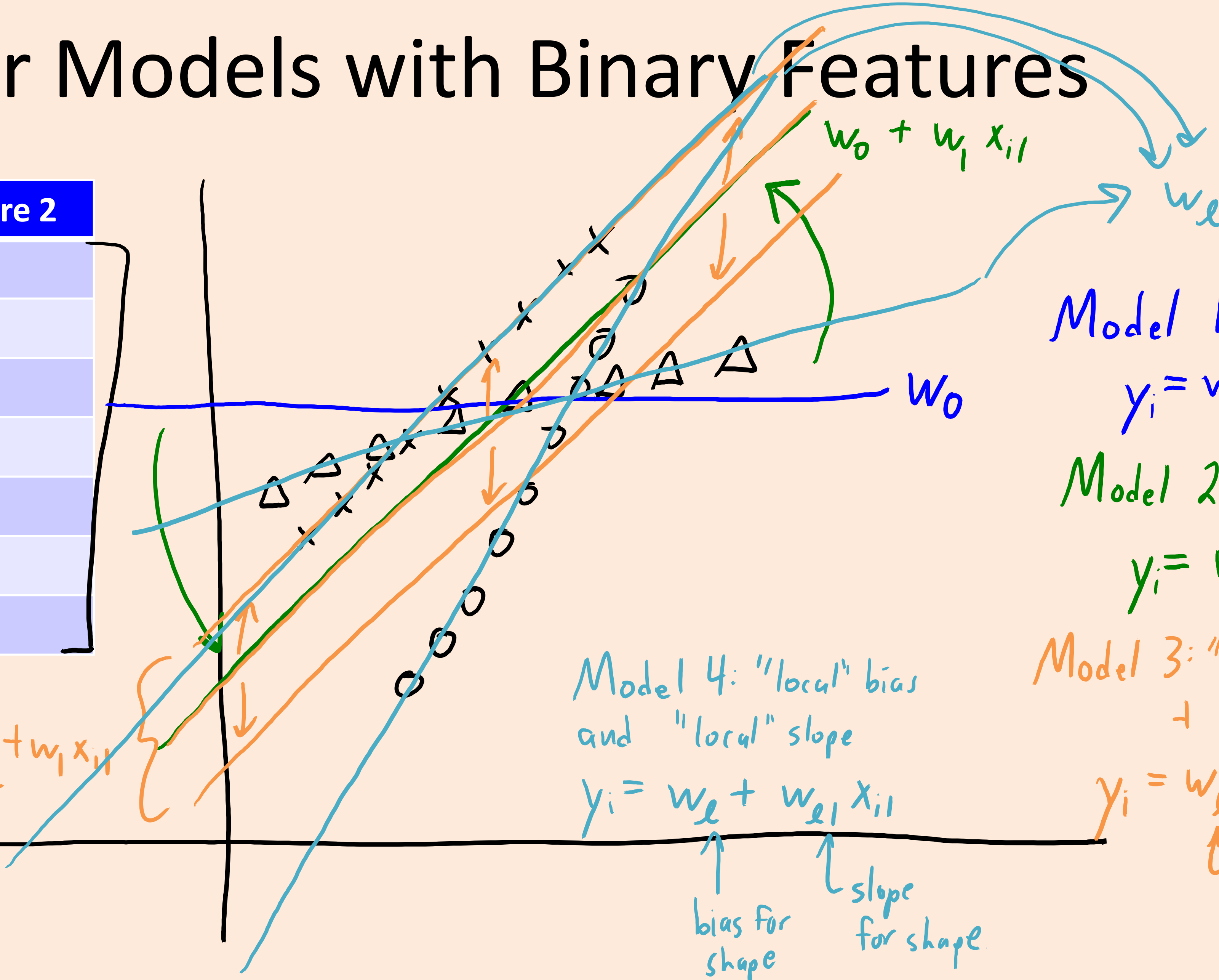
$y_i = w_\ell + w_1 x_{i1}$

shape

Linear Models with Binary Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only <u>bias</u>

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

$y_i = w_\ell + w_1 x_{i1}$

shape

Model 4: "local" bias and "local" slope

$y_i = w_\ell + w_{\ell 1} x_{i1}$

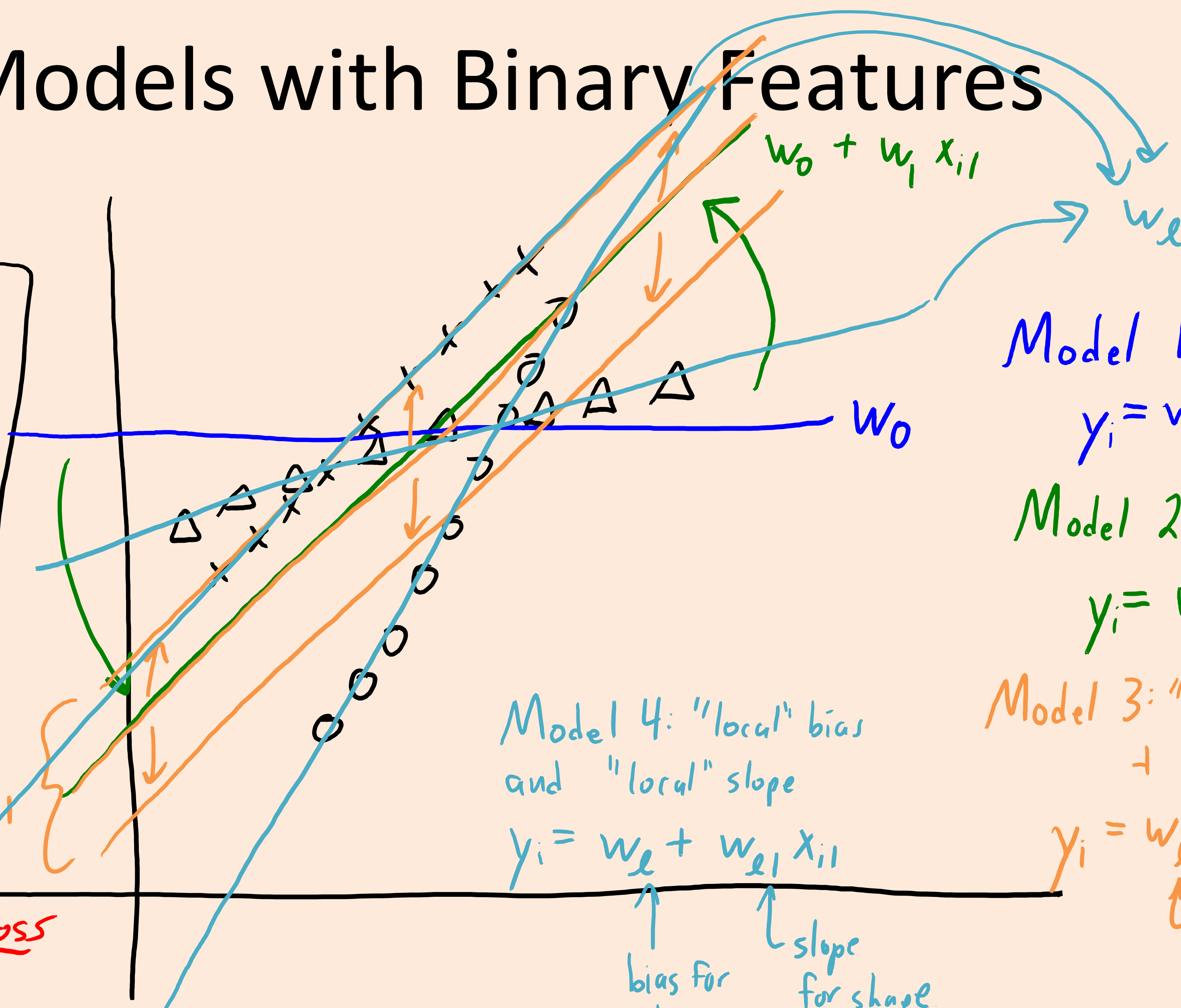
bias for shape

slope for shape

Linear Models with Binary Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

$y_i = w_\ell + w_1 x_{i1}$

shape

Model 4: "local" bias and "local" slope

$y_i = w_\ell + w_{\ell 1} x_{i1}$

bias for shape

slope for shape

Could also share information across categories with global bias slope:

$y_i = w_0 + w_1 x_{i1} + w_\ell + w_{\ell 1} x_{i\ell}$

Global and Local Features for Domain Adaptation

- Suppose you want to solve a classification task, where you have very little labeled data from your domain.
- But you have access to a huge dataset with the same labels, from a different domain.
- Example:
  - You want to label POS tags in medical articles, and pay a few $$$ to label some.
  - You have access the thousands of examples of Wall Street Journal POS labels.
- Domain adaptation: using data from different domain to help.

Global and Local Features for Domain Adaptation

- "Frustratingly easy domain adaptation":
  - Use "global" features across the domains, and "local" features for each domain.
  - "Global" features let you learn patterns that occur across domains.
    - Leads to sensible predictions for new domains without any data.
  - "Local" features let you learn patterns specific to each domain.
    - Improves accuracy on particular domains where you have more data.
  - For linear classifiers this would look like:

$$\hat{y}_i = \text{sign}(\underbrace{w_g^T x_{ig}}_{\text{features used across domains}} + \underbrace{w_d^T x_{id}}_{\text{features/weights } \underline{\text{specific}} \text{ to domain}})$$

FFT implementation of convolution

- Convolutions can be implemented using fast Fourier transform:
  - Take FFT of image and filter, multiply elementwise, and take inverse FFT.

- It has faster asymptotic running time but there are some catches:
  - You need to be using periodic boundary conditions for the convolution.
  - Constants matter: it may not be faster in practice.
    - Especially compared to using GPUs to do the convolution in hardware.
  - The gains are largest for larger filters (compared to the image size).