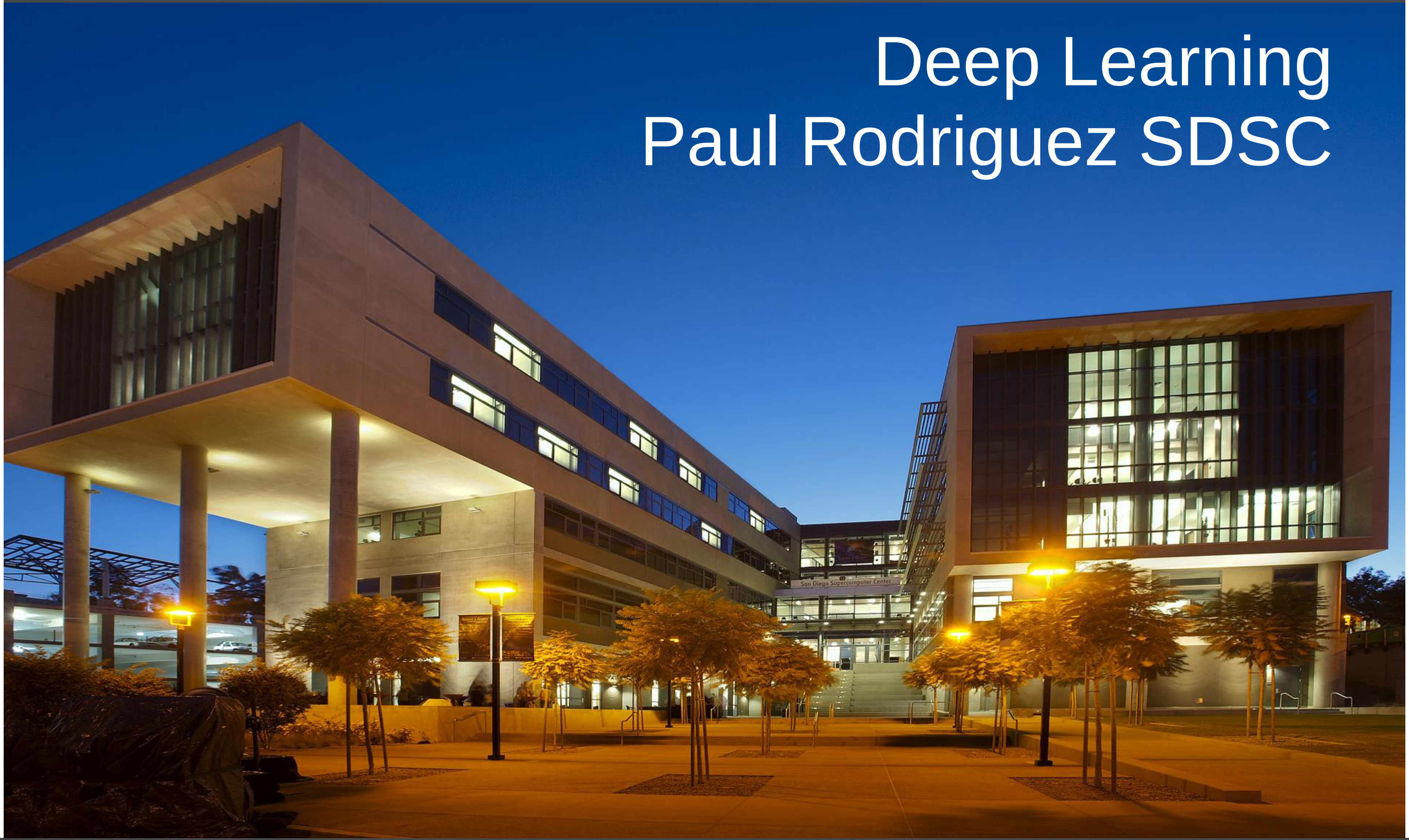
Deep Learning
Paul Rodriguez SDSC



Outline

I. What is Deep Learning
II. Neural Networks
III. Convolution Neural Networks
IV. Tutorial

Deep Learning

- **3 characterizations:**

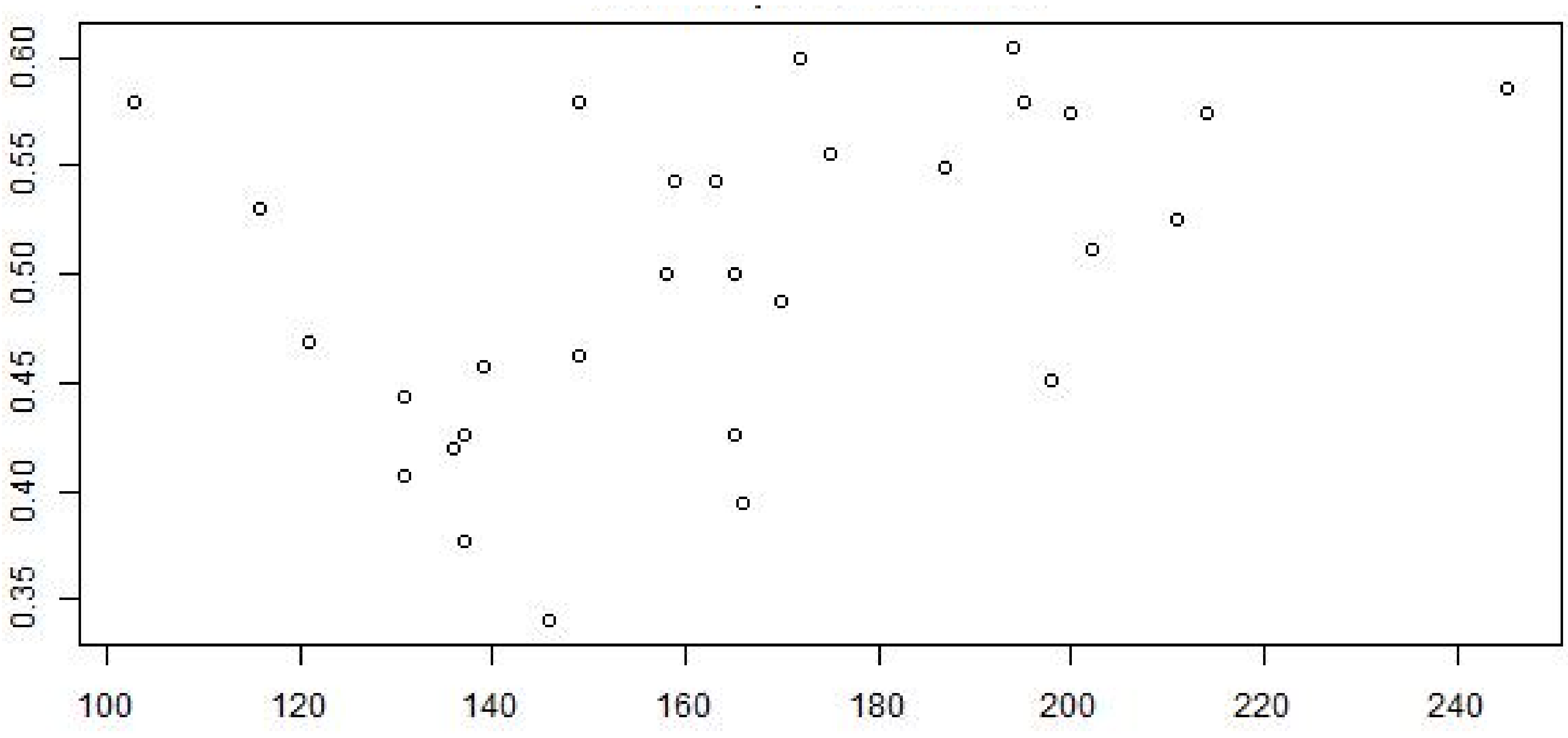
1. Learning complicated interactions about input
2. Learning complex feature transformations
3. Using neural networks with many layers

Explanation Strategy: Start with linear regression and go deep

A data example:Home Runs and W-L percent

W-L %

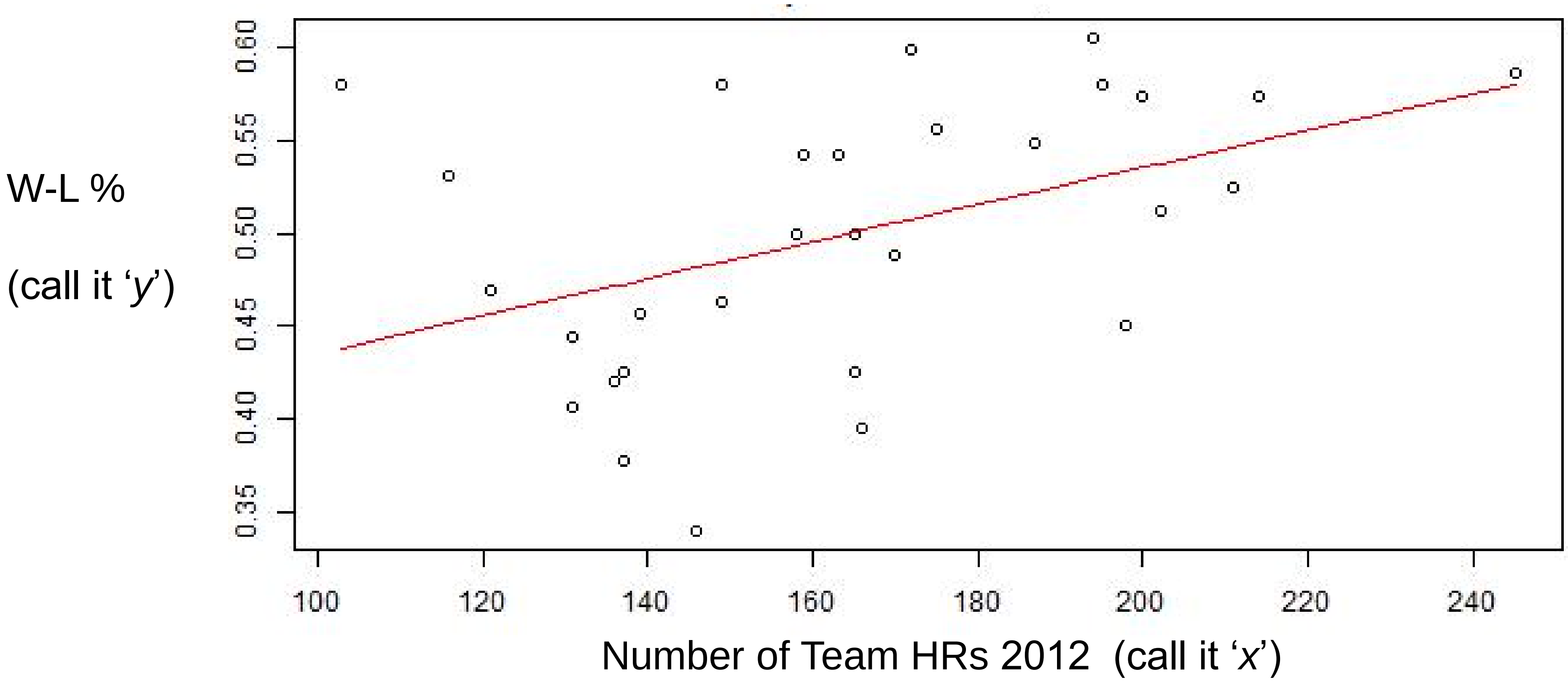
(call it ‘*y*’)



Number of Team HRs 2012 (call it ‘*x*’)

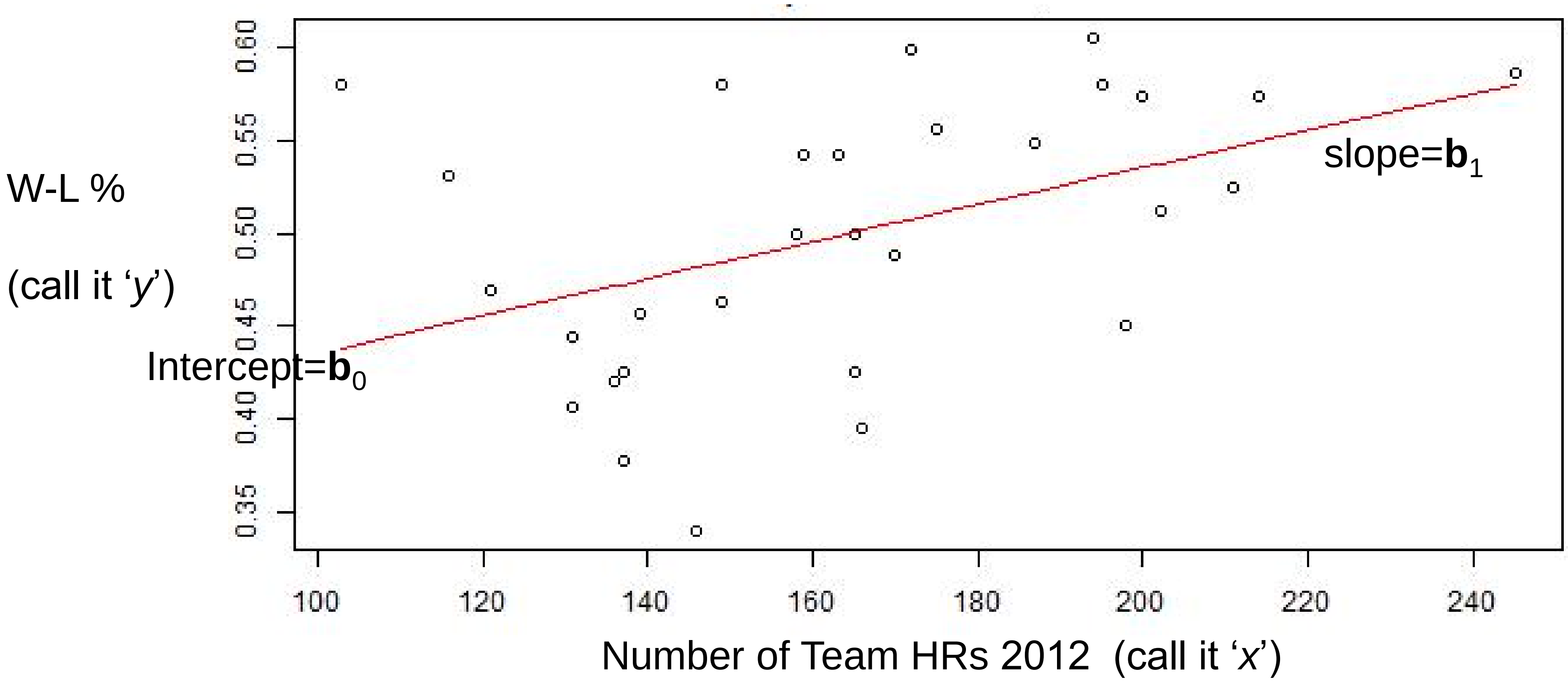
Recall Linear Regression is Fitting a Line

the Model: $y = f(x, b) = b_o * 1 + b_1 * x$



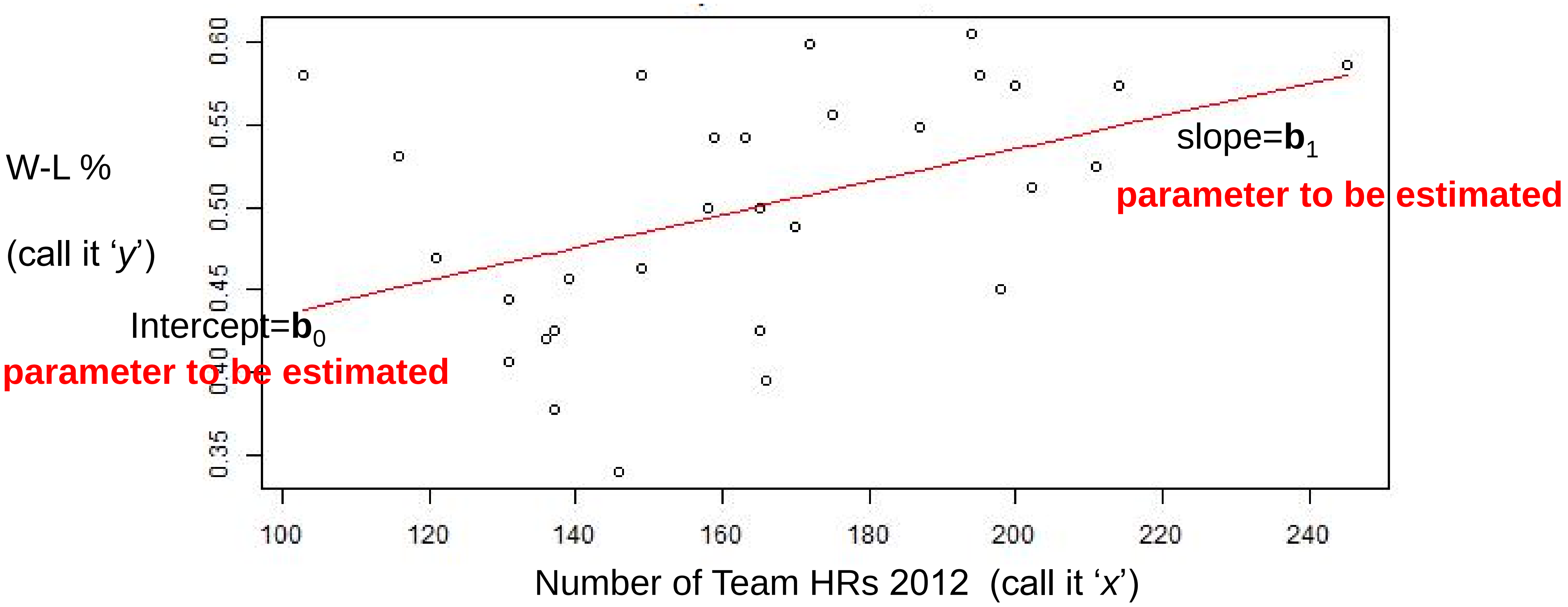
Recall Linear Regression is Fitting a Line

the Model: $y = f(x, b) = b_o * 1 + b_1 * x$



Recall Linear Regression is Fitting a Line

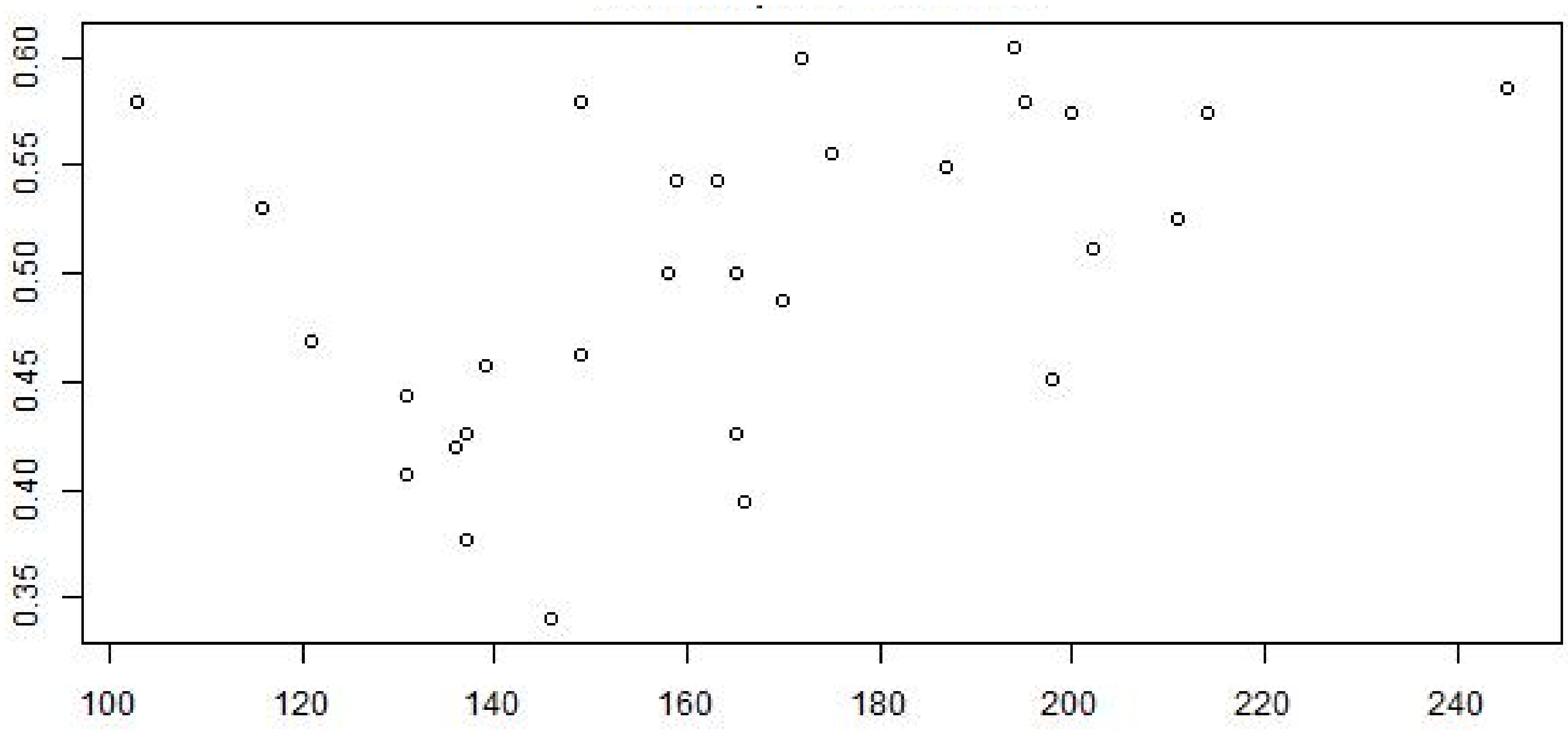
the Model: $y = f(x, b) = b_o * 1 + b_1 * x$



Can we just classify winners vs losers based on home runs?

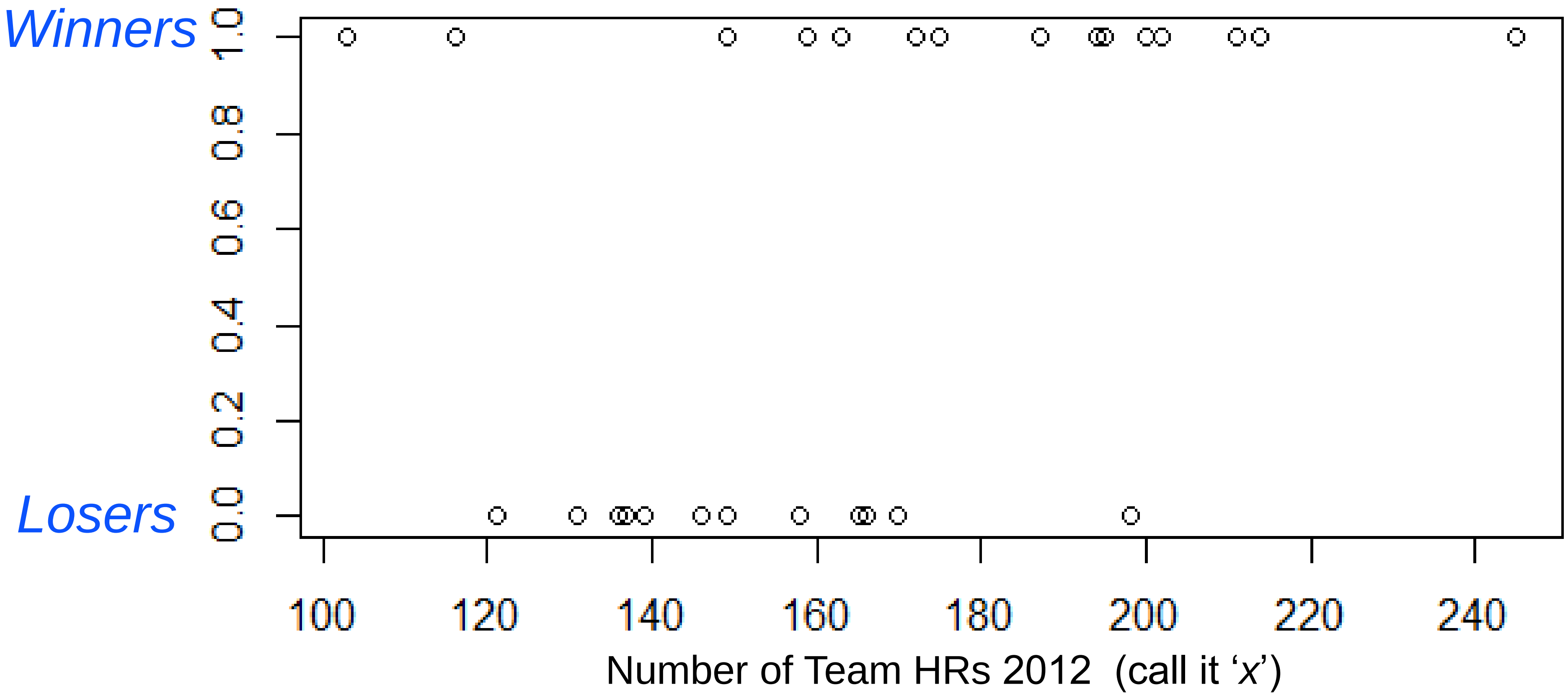
W-L %

(call it ‘*y*’)



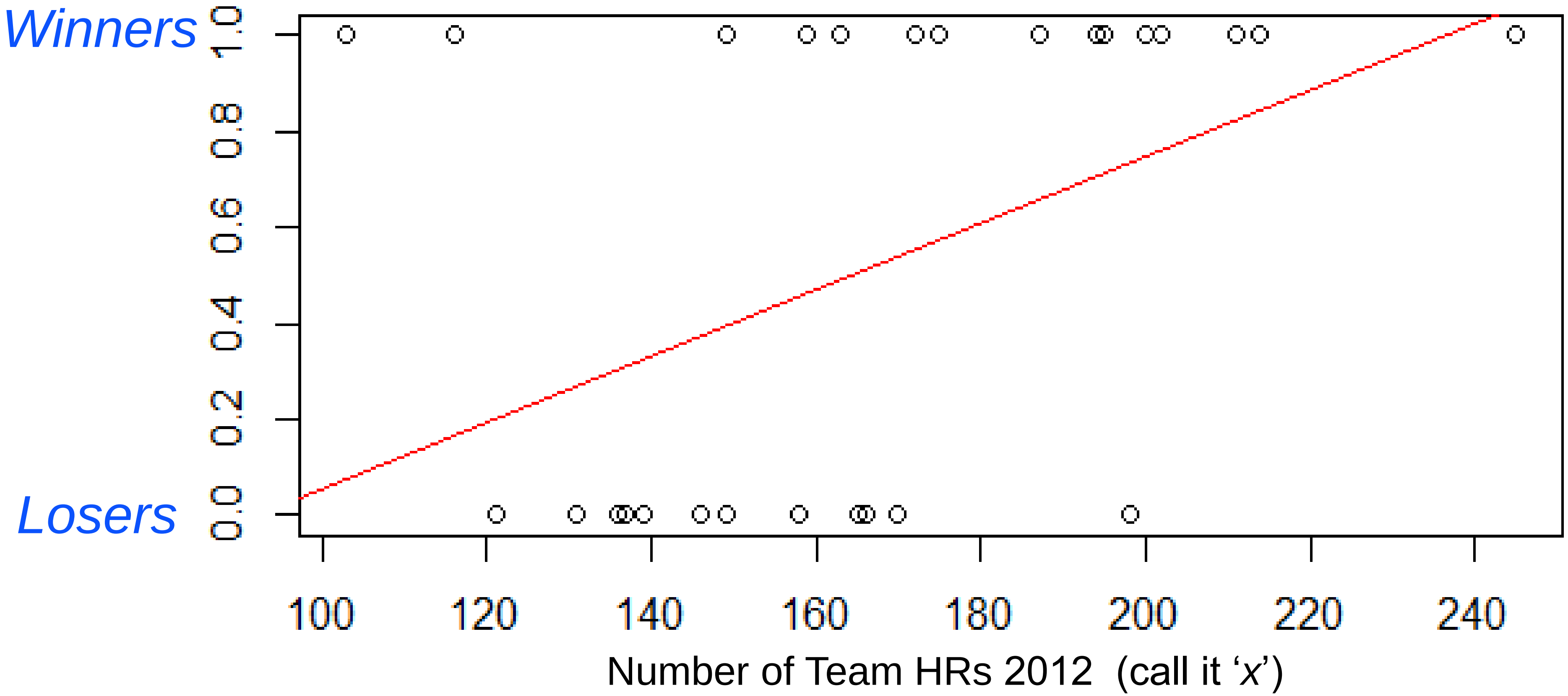
Number of Team HRs 2012 (call it ‘*x*’)

Classification uses labelled outcomes



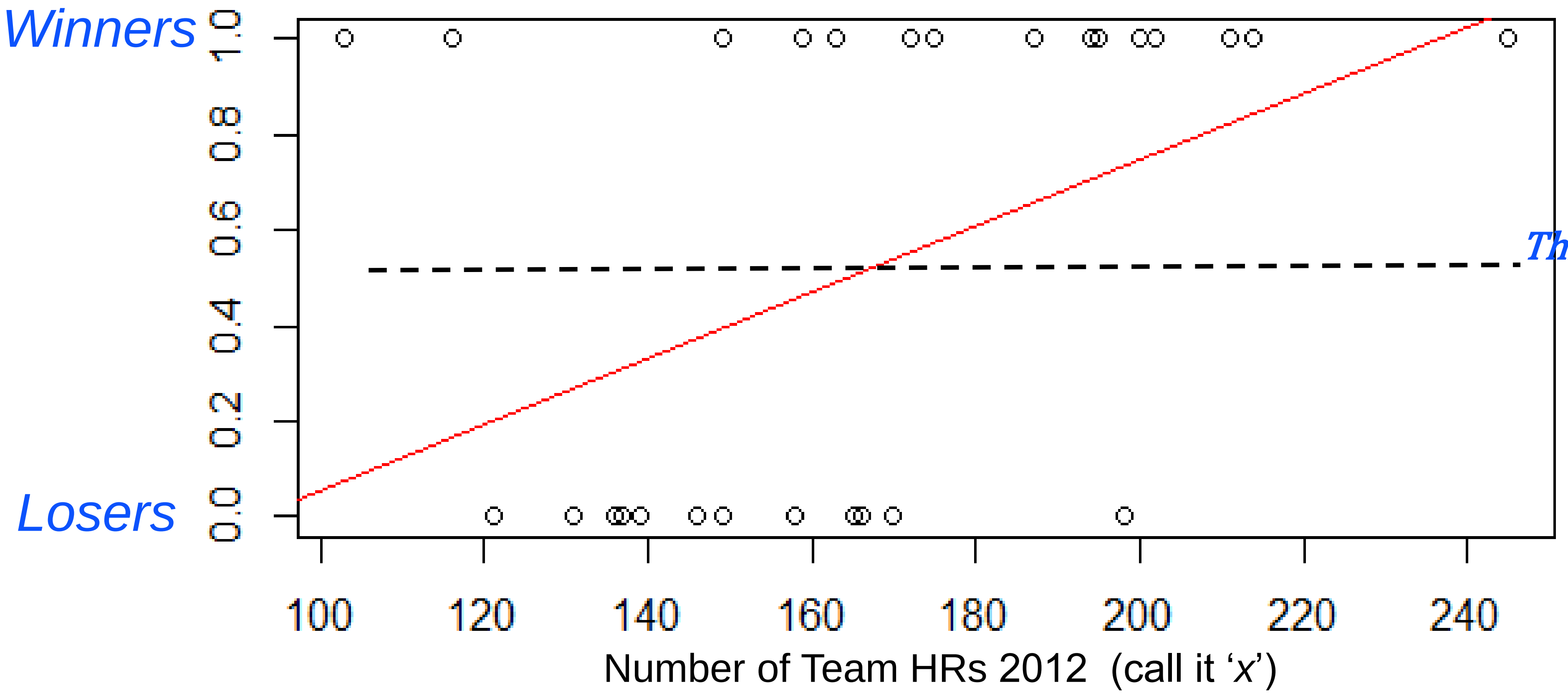
Can do this: fit a line with same model

$$\textbf{the Model}: \; y = f(x, b) = b_o * 1 + b_1 * x$$



Can do this: fit a line with same model

the Model: $y = f(x, b) = b_o * 1 + b_1 * x$

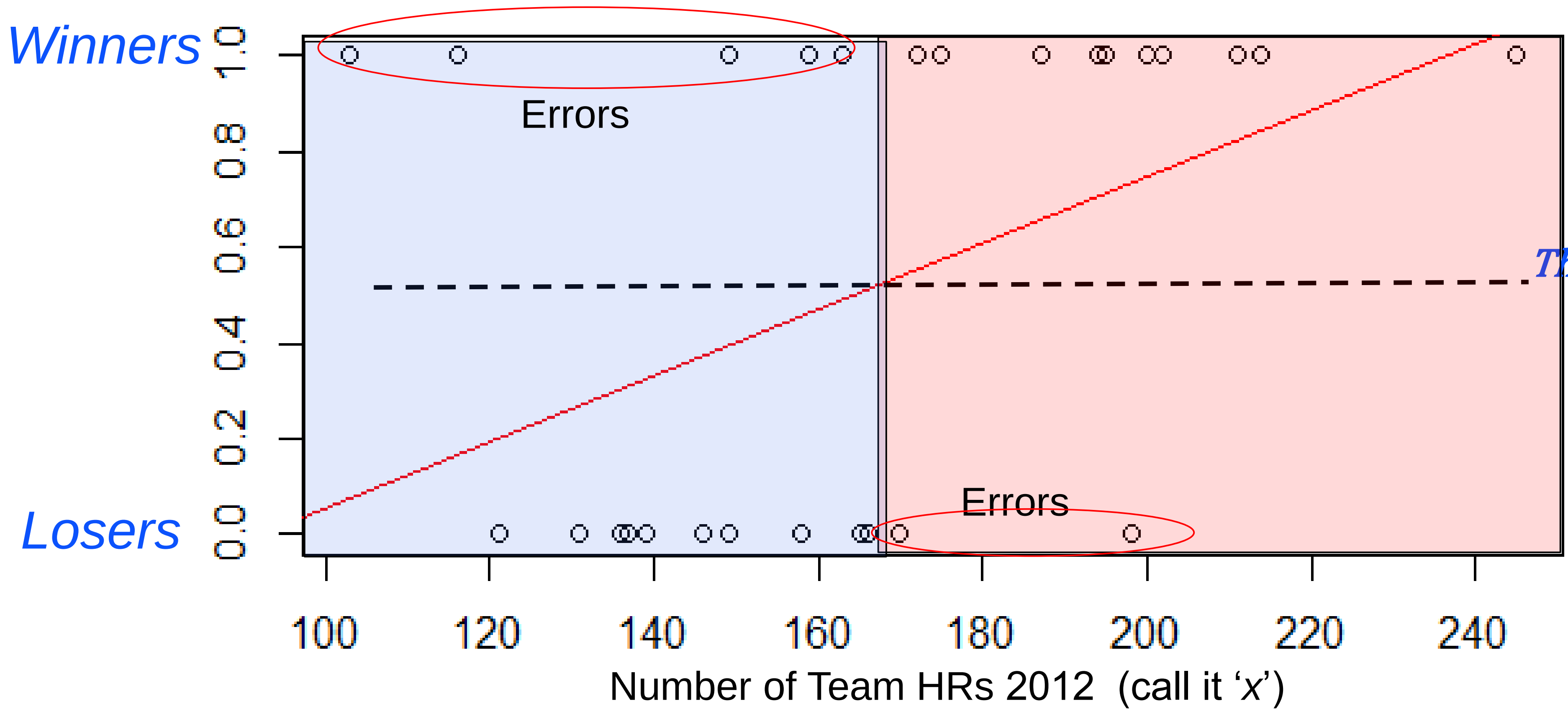


Use threshold to label prediction:

Threshold:
for $y > 0.5$, classify x as Winner

Can do this: fit a line with same model

the Model: $y = f(x, b) = b_o * 1 + b_1 * x$



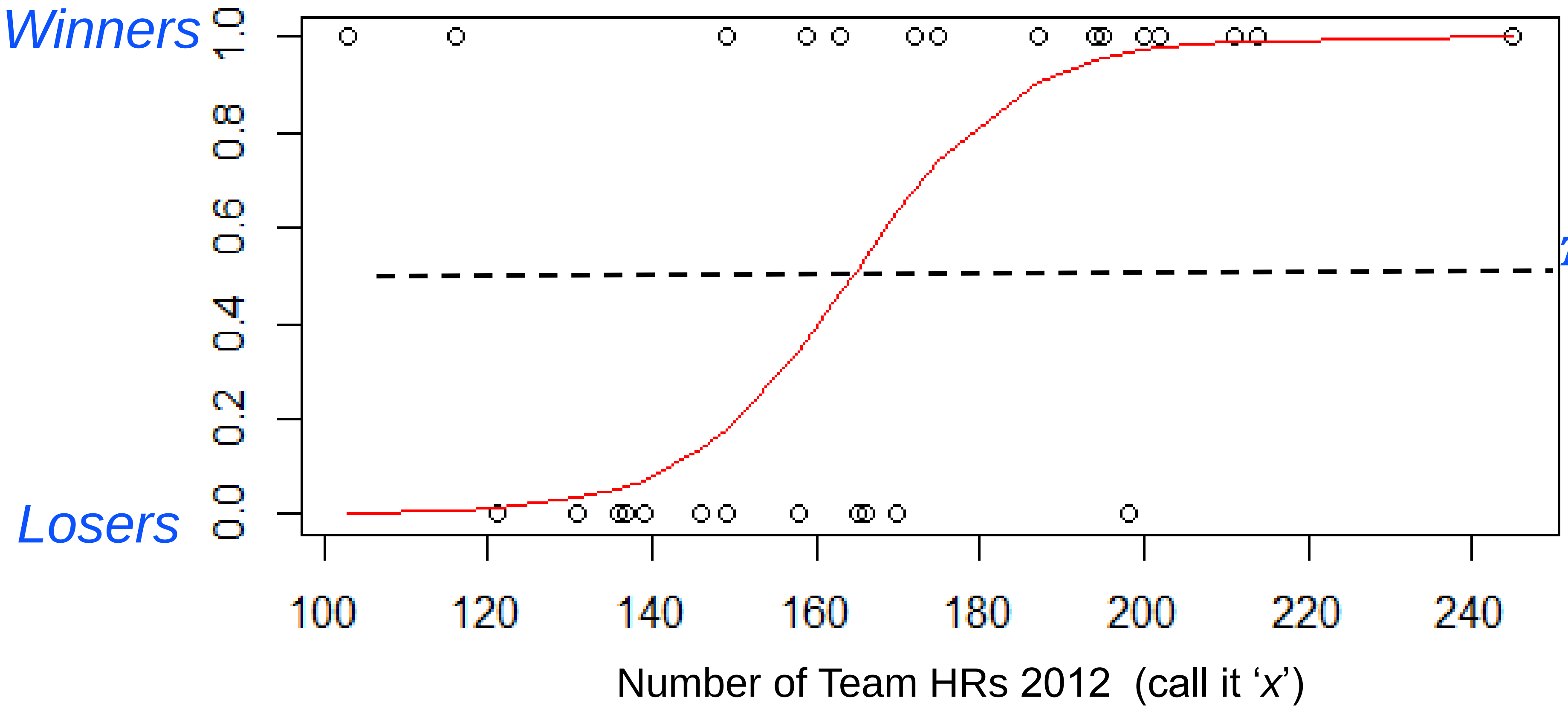
Use threshold to label prediction:

Threshold:
for $y > 0.5$, classify x as Winner

Divides X into Winner/Loser regions

Can do better: fit a nonlinear function

the Model: $y = f(x, b) = 1/(1 + \exp[-(b_o * 1 + b_1 * x)]$



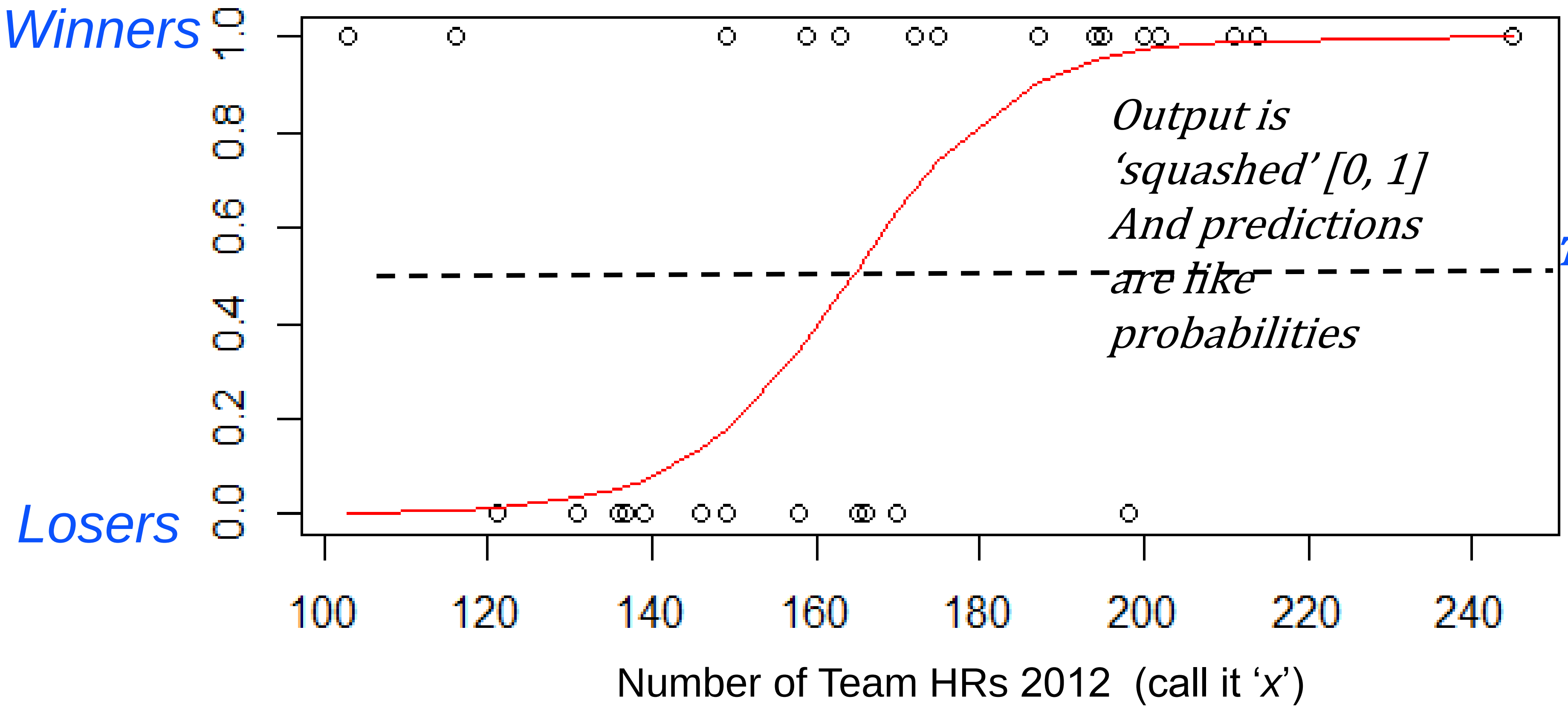
Use threshold to label prediction:

Threshold:

for $y > 0.5$, classify x as Winner

Can do better: fit a nonlinear function

the Model: $y = f(x, b) = 1/(1 + \exp[-(b_o * 1 + b_1 * x)]$



Use threshold to label prediction:

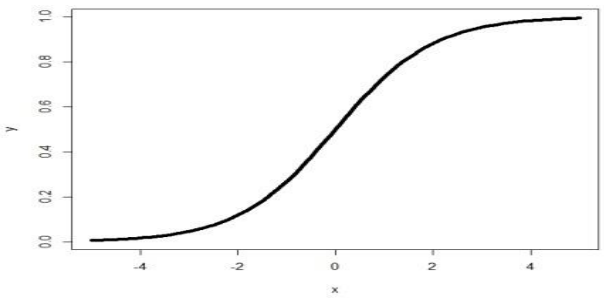
Threshold:

for $y > 0.5$, classify x as Winner

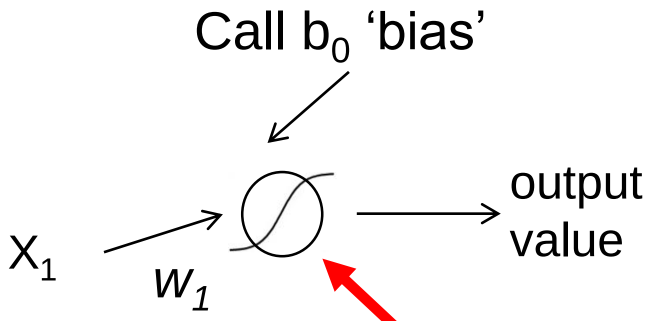
Logistic to Neural Network model

- $\mathbf{y} = b_o * 1 + b_1 * x$ **= B*X**

- **Squash** $b_o * 1 + b_1 * x$ **to 0,1 range using Logistic Function [1/(1+exp(-BX)]:**



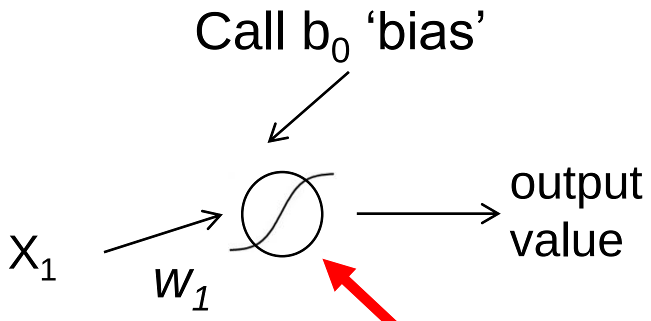
Logistic Regression as 1 node network



Note:
call betas
weights

logistic function to transform input to output – call it the 'activation' function

Logistic Regression as 1 node network

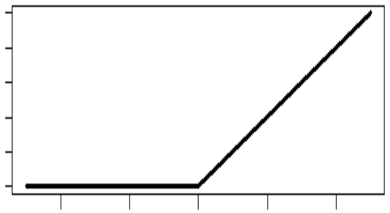


Note:
call betas
weights

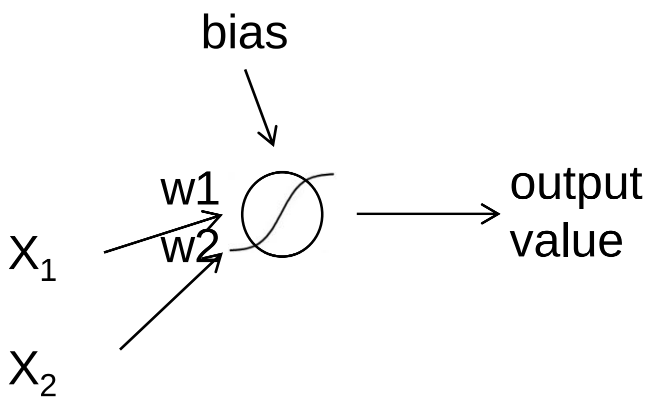
logistic function to transform input to output – call it the 'activation' function

Note: other activations are possible,

RELU (rectified linear unit)



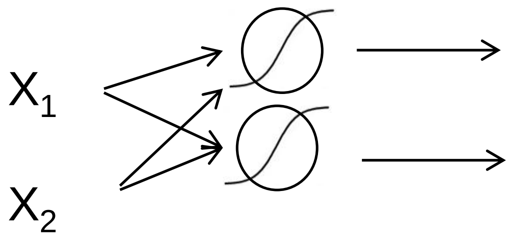
Next step: More general networks



Add input variables

More general networks

(assume bias present)

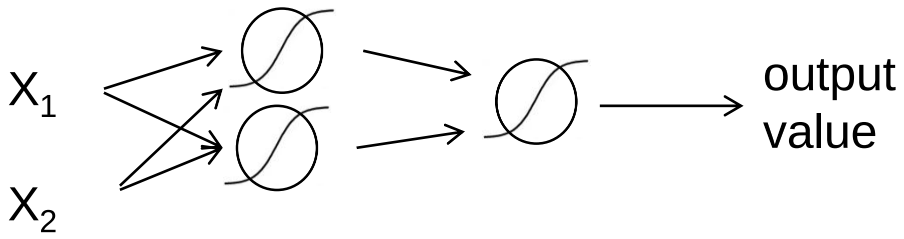


Add input variables

Add logistic transformations …

More general networks

(assume bias)



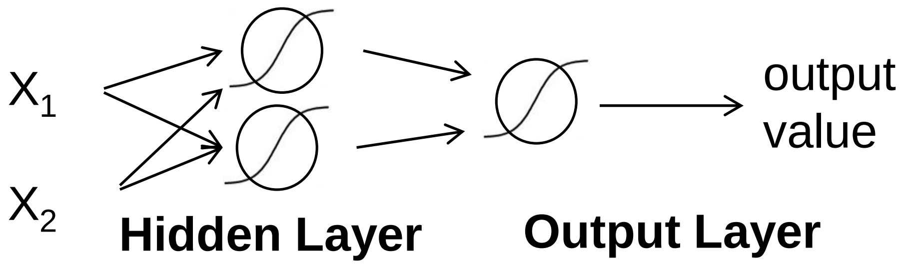
Combine transformations!

Add input variables

Add logistic transformations …

More general networks

(assume bias)



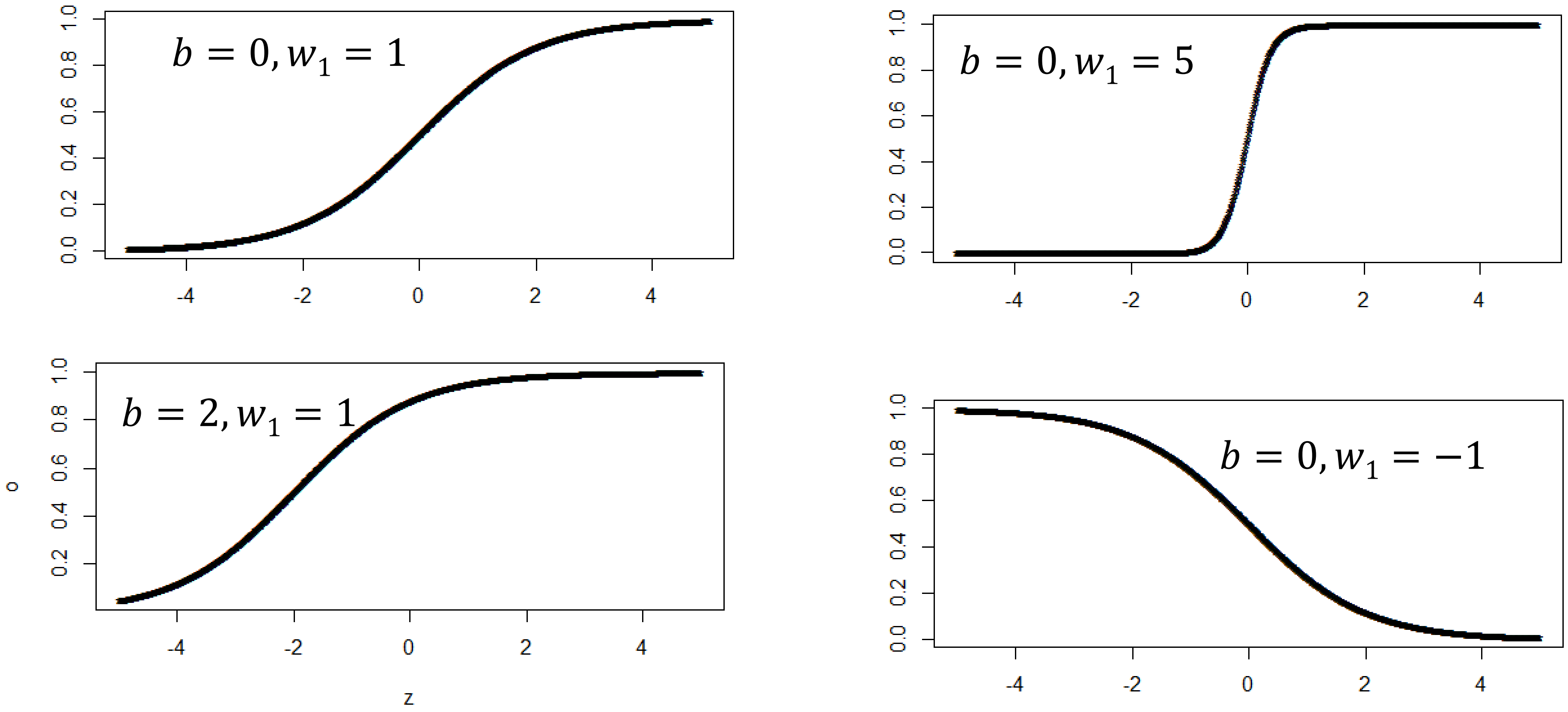
Combine transformations!

Add input variables

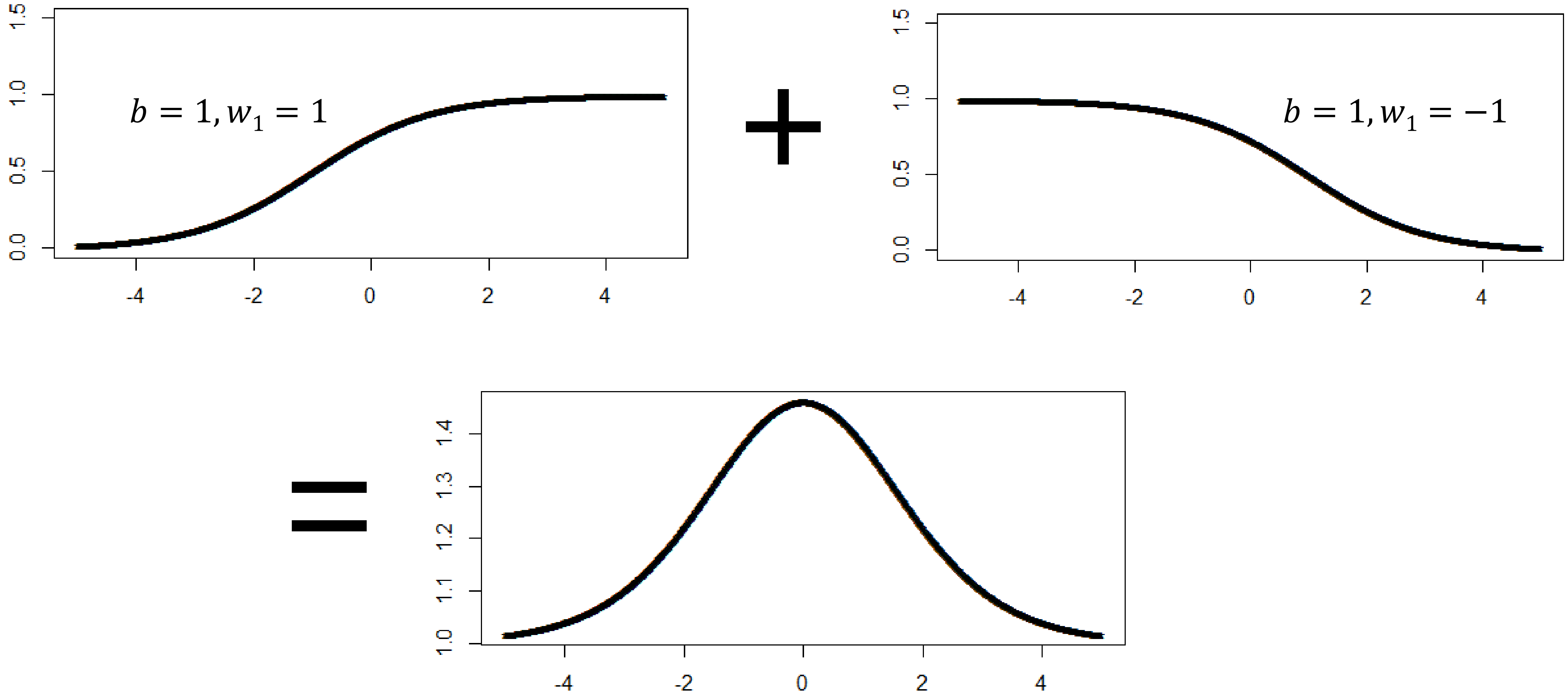
Add logistic transformations …

Logistic function w/various weights

$$for\ y = 1/(1+exp(-(b+w_1 * x))$$

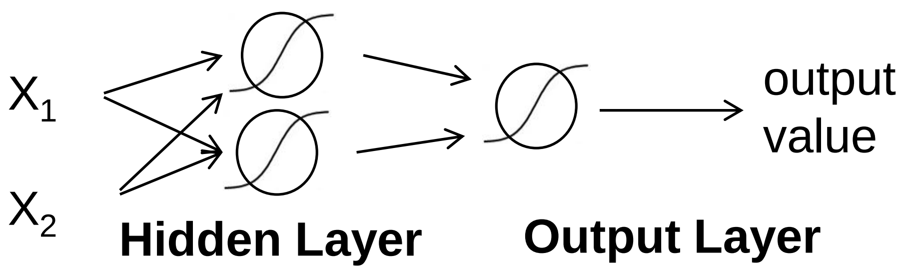


So combinations are highly flexible and nonlinear



But parameter fitting is harder too

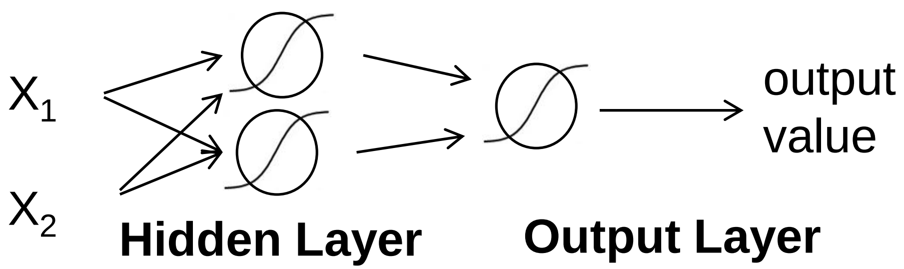
(assume bias present)



Error signals only known for output layer, so they have to ‘back-propagate’ to hidden layers

But parameter fitting is harder too

(assume bias present)

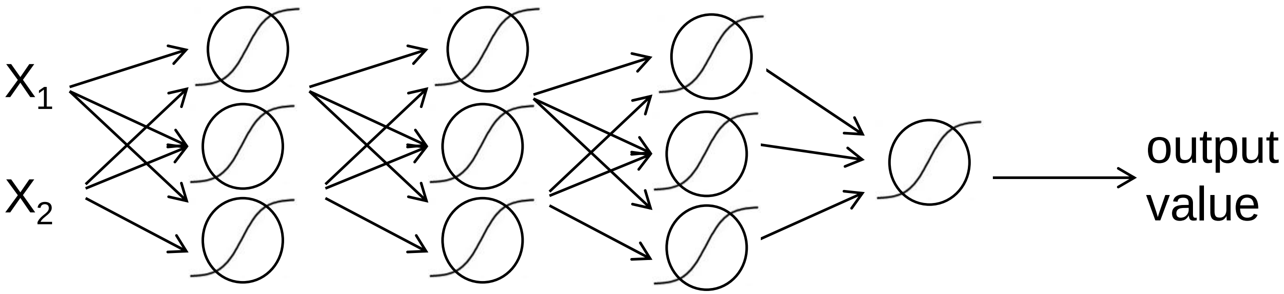


Error signals only known for output layer, so they have to ‘back-propagate’ to hidden layers

Use derivatives and chain rules and iterate over training data (stochastic gradient descent) to adjust weights

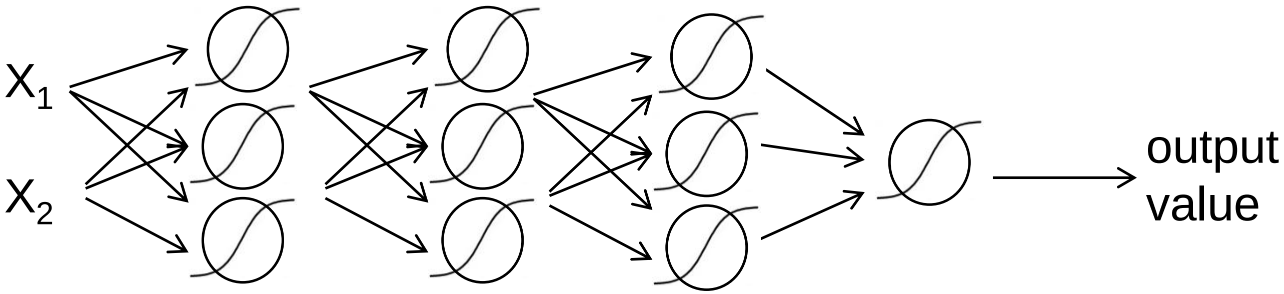
Why stop at 1 hidden layer?

- More hidden layers => More varied features, or ‘Deep’ Learning



Train with Care

- More hidden layers => More varied features, or ‘Deep’ Learning



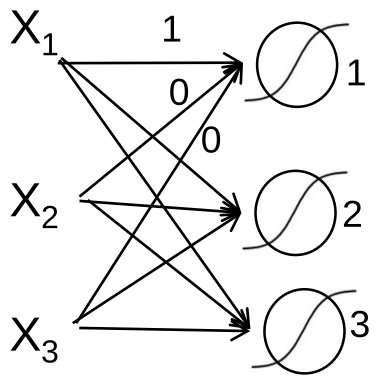
Many more parameters, and error signal at final output layer gets drowned out at lower layers-

but penalizing weight sizes, varied activation functions, and more data help!

Feature Transformations, Projections, and Convolutions

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume b_0=0, assume all X normalized between 0 and 1)



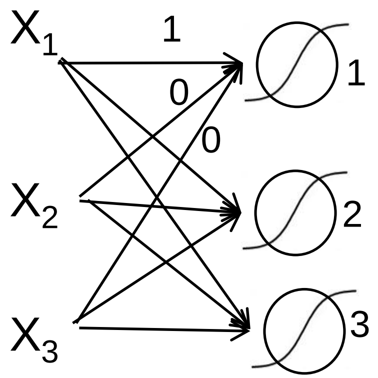
Call the connection parameters 'weights'.

For node 1 let $[w_1\ w_2\ w_3] = [1\ 0\ 0]$

What feature transformation is that?

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume b_0=0, assume all X normalized between 0 and 1)



Call the connection parameters 'weights'.

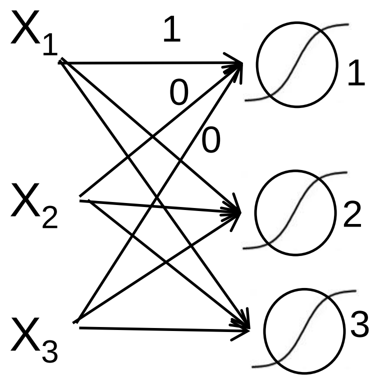
For node 1 let [w_1 w_2 w_3] = [1 0 0]

What feature transformation is that?

Informally, squash X1 and ignore X2,X3

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1\ w_2\ w_3\ ] = [1\ 0\ 0]$

For node 2 let $[w_1\ w_2\ w_3\ ] = [0\ 1\ 0]$

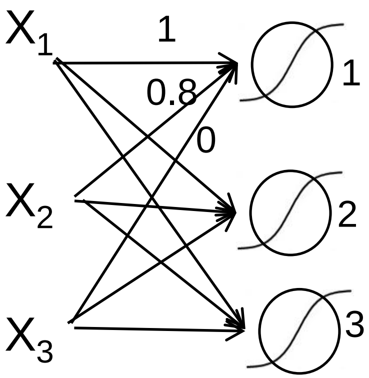
For node 3 let $[w_1\ w_2\ w_3\ ] = [0\ 0\ 1]$

What feature transformation are these together?

Informally, squash 3D to another 3D space

A Factor Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1\ w_2\ w_3\ ] = [1 \quad 0.8 \quad 0]$

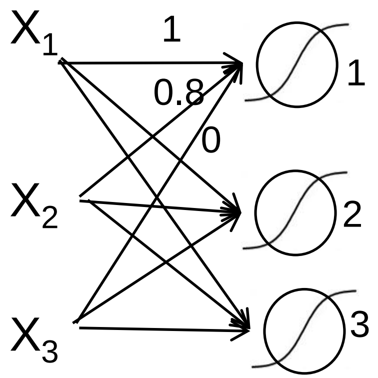
For node 2 let $[w_1\ w_2\ w_3\ ] = [-0.8 \quad 1 \quad 0]$

For node 3 let $[w_1\ w_2\ w_3\ ] = [0 \quad 0 \quad 0]$

What feature transformation are these together?

A Factor Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1\ w_2\ w_3\ ] = [1 \quad 0.8 \quad 0]$

For node 2 let $[w_1\ w_2\ w_3\ ] = [-0.8 \quad 1 \quad 0]$

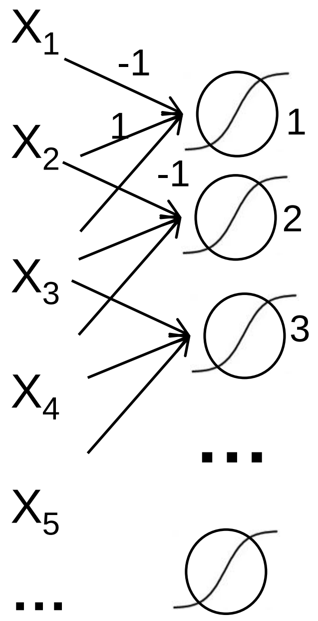
For node 3 let $[w_1\ w_2\ w_3\ ] = [0 \quad 0 \quad 0]$

What feature transformation are these together?

Informally, like projection onto 2 orthogonal dimensions (recall PCA example on Athletes Height and Weight!)

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)

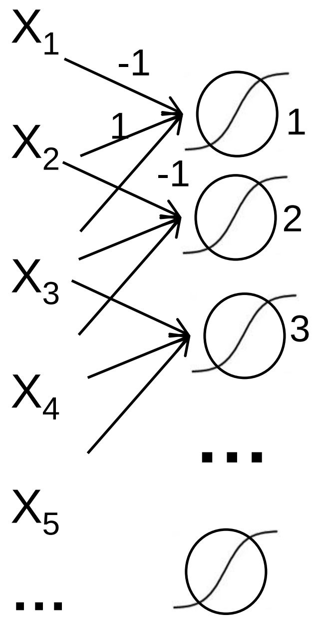


For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

What is the node 1 doing?
(assuming W are just +/- 1)

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)



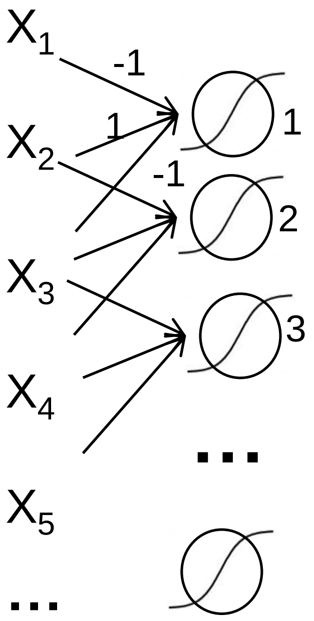
For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

What is the node 1 doing?
(assuming x are just +/- 1)

Informally, node 1 has max activation for a 'spike', e.g. when X_2 is positive and X_1, X_3 are negative

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)



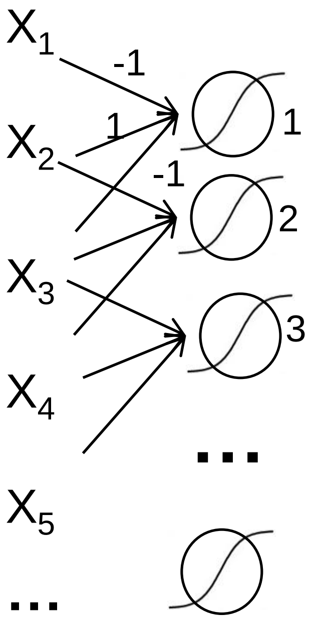
For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

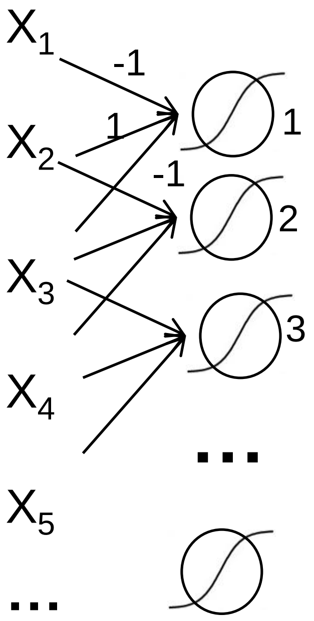
What is the hidden layer doing?

Informally, looking for a spike everywhere.

This is essentially a convolution operator, where W is the kernel.

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

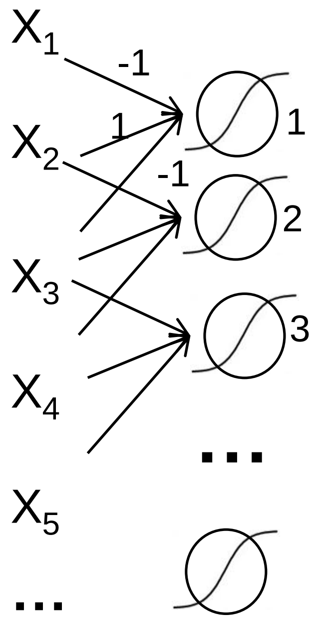
Informally, looking for a spike everywhere.

This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding* W across input

A Filter

Many X input, but only 3 connections to each hidden node
(a local connectivity pattern, aka receptive field)



For node 1 let W=[w_1 w_2 w_3] = [-1 1 -1]

For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere.

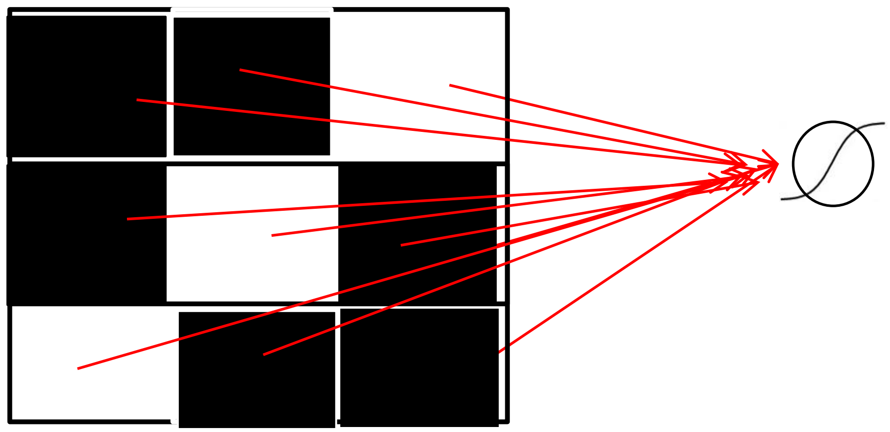
This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding* W across input

Note: if we take max activation across nodes ('Max Pool') then it's like looking for a spike *anywhere*.

2D Convolution

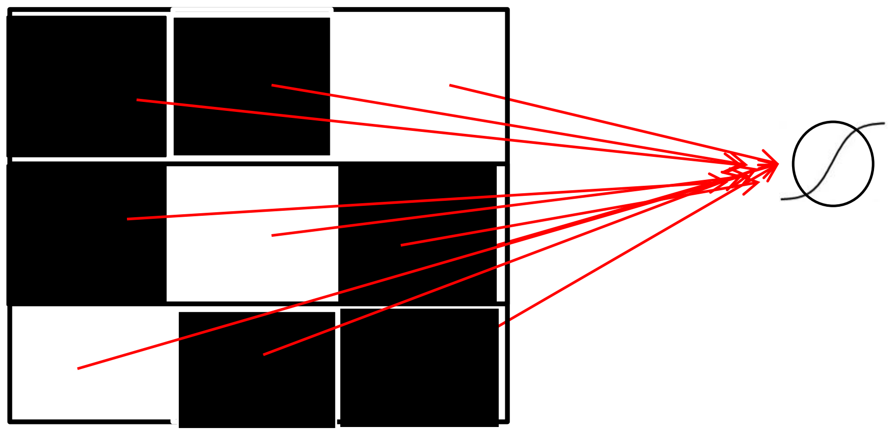
Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node



What W matrix would ‘activate’ for a upward-toward-left diagonal line?

2D Convolution

Now let input be a 2D binarized 3x3 matrix fully connected to 1 node



(black= -1 white=1)

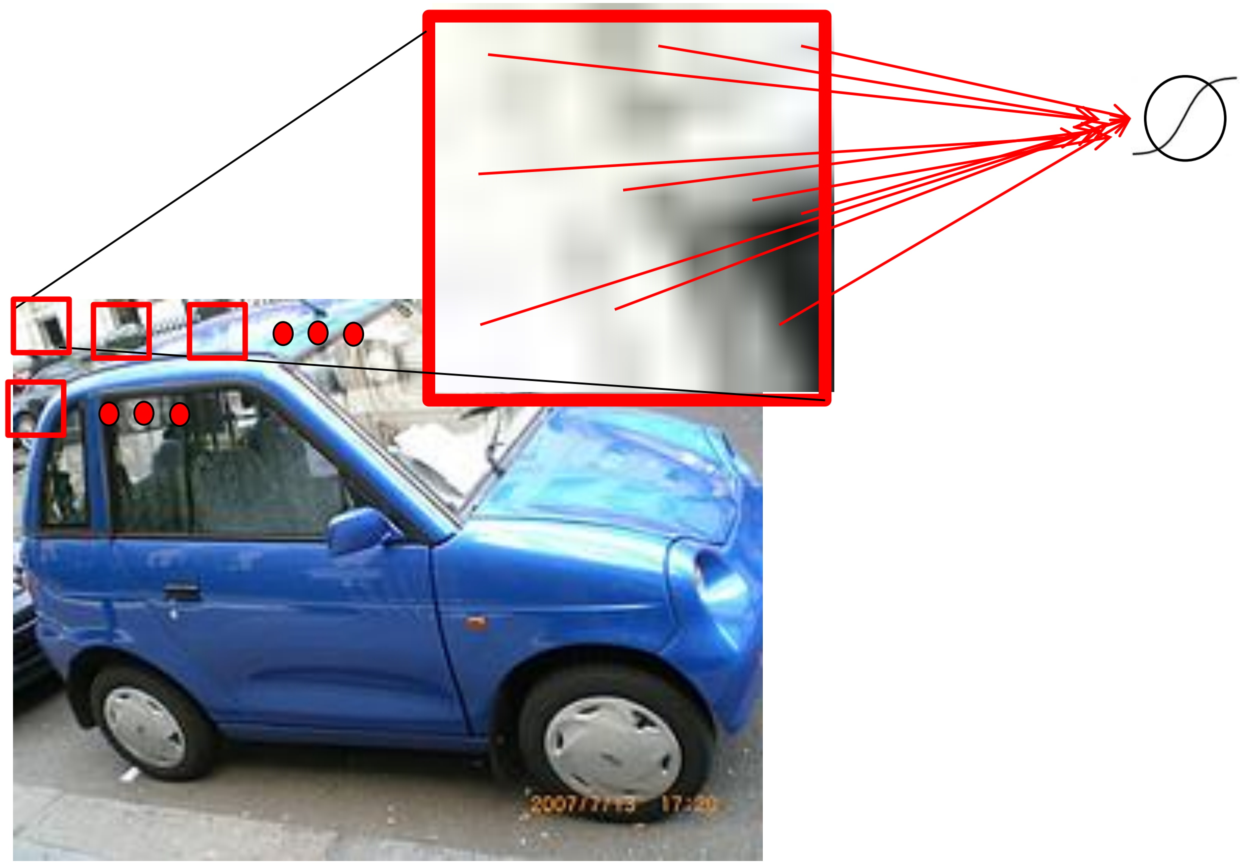
What W matrix would 'activate' for a upward-toward-left diagonal line?

How about:

$$W = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

2D Convolution

For full image, 1 filter is applied to 1 region in 1 color channel at a time, and then slid across regions (or done in parallel with shared weights) and produces 1 new 2D image (hidden) layer



Convolution Layer parameters:

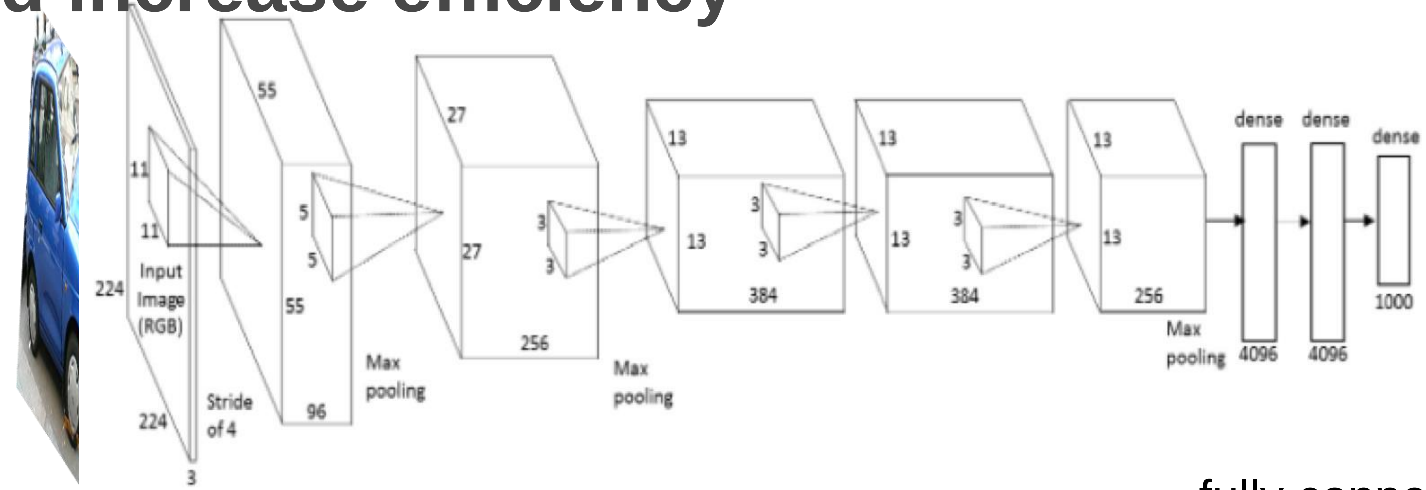
- filter size depends on input:
  - smaller filters for smaller details
  - 2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
  - smaller better but less efficient
- number of filters
  - depends on task
  - each filter is a new 2D layer

Convolution Network :

many layers and architecture options

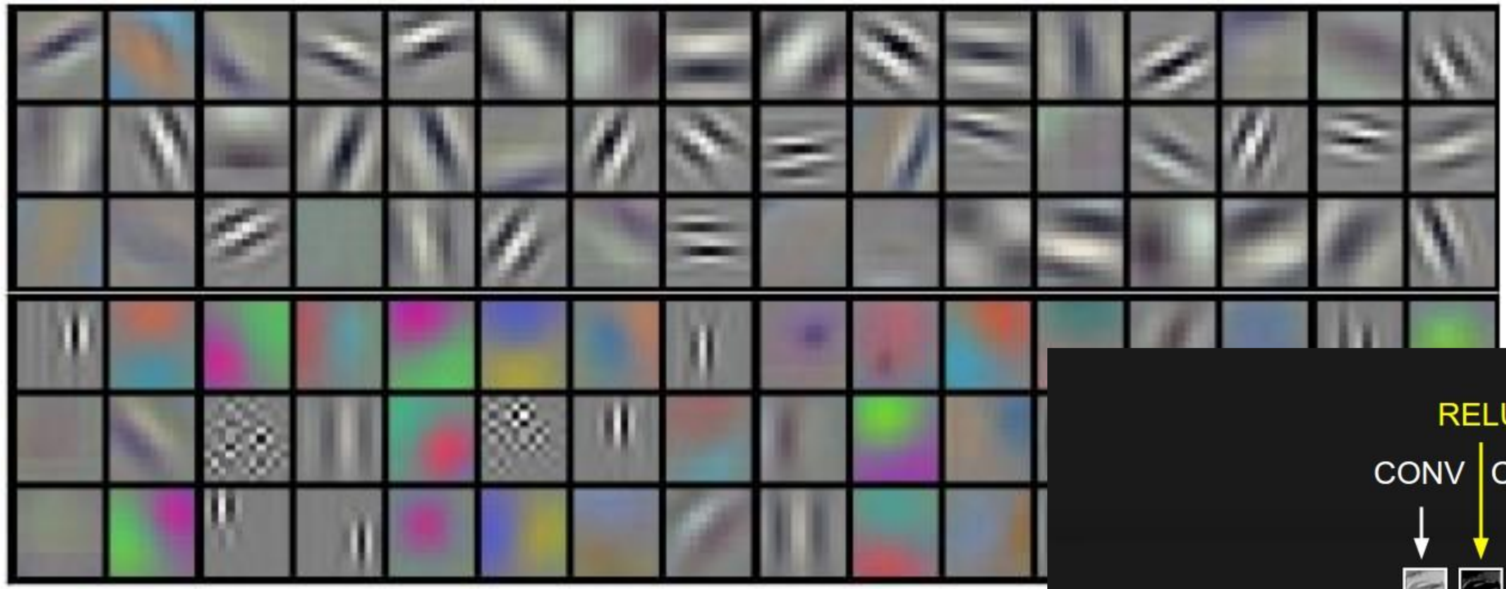
Large Scale Versions

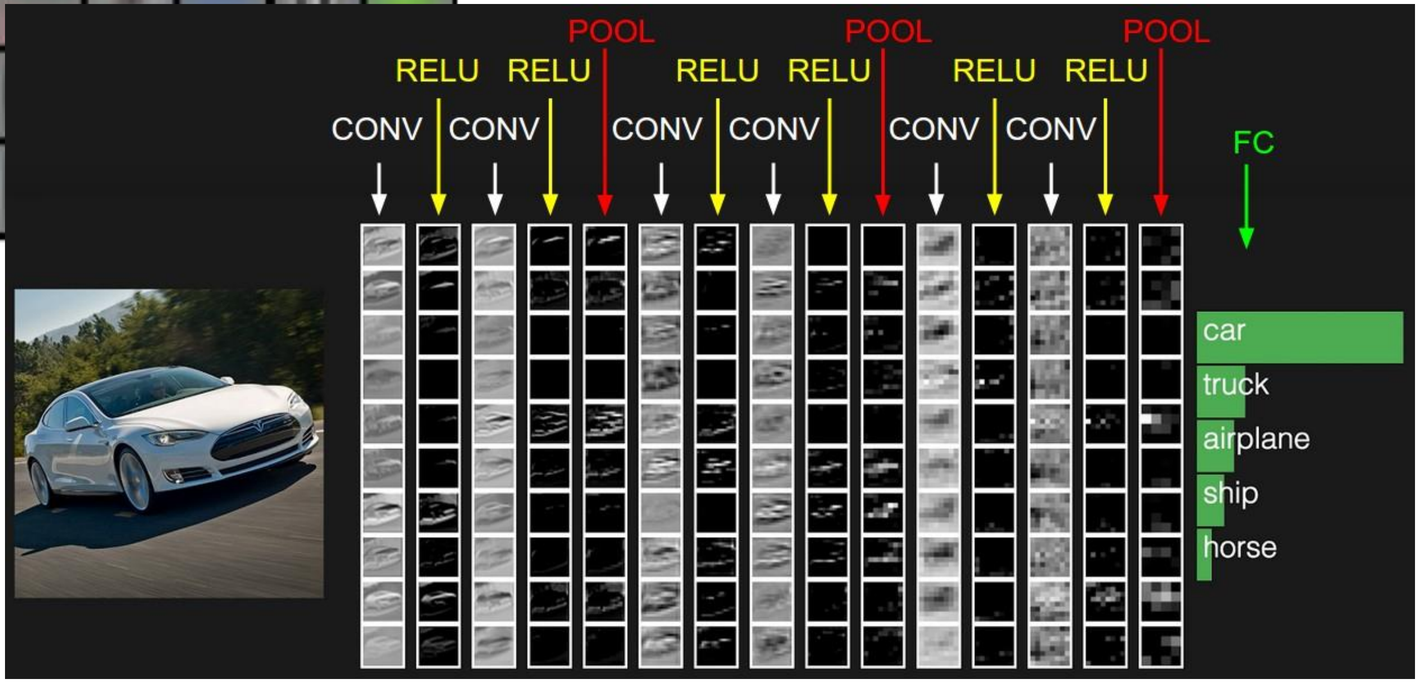
- **Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)**
- **Need large amounts of data and many heuristics to avoid overfitting and increase efficiency**



Convolution layer followed by RELU layer (rectified linear activation units instead of logistic function) followed by Max Pooling layer (over 2D regions with sliding) ➡ fully connected layers and output layers (standard neural network layers)

What Learned Convolutions Look Like





Summarizing Deep Layers

- **Hidden layers transform input into new features:**
  - Feature can be highly nonlinear
  - Features as a new space of input data
  - Features as projection onto lower dimensions (compression)
  - Features as filters, which can be used for convolution
- **But also:**
  - Many algorithm parameters
  - Many weight parameters
  - Many options for stacking layers

Feature Coding vs Discovery

- **Some problems can work with judicious feature selection (e.g. Haar cascades work well for face detection)**

- **Edge detection functions can be used as input for non-neural network classifiers (e.g. histogram of gradients with support vector machines)**

- **Building features is hard, and large classification problems can benefit from common features, so Convolution networks are used to discover features for multiclass outputs**

References

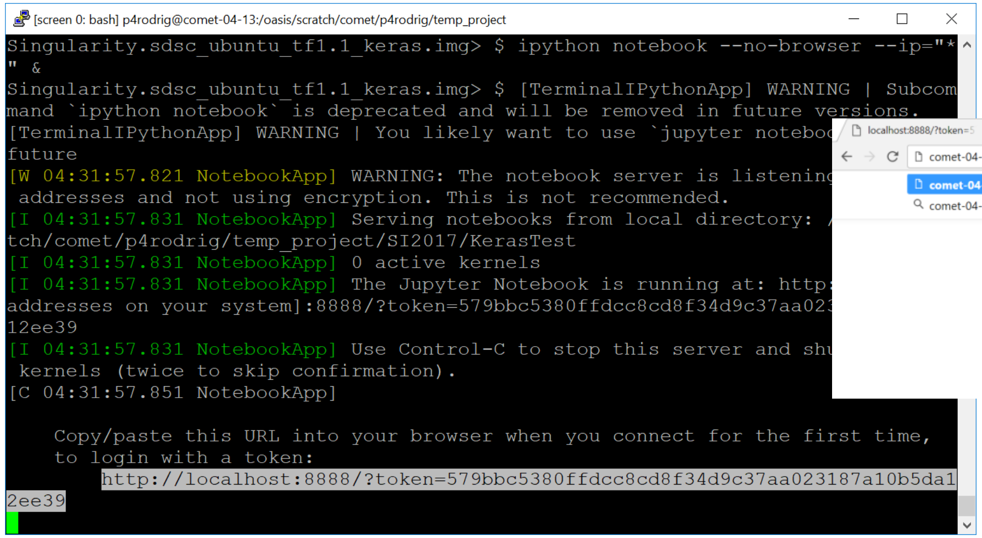
- **Book: https://mitpress.mit.edu/books/deep-learning**
- **Documentation: https://keras.io/**
- **Tutorials I used (borrowed):**
  - http://cs231n.github.io/convolutional-networks/
  - https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59
  - https://github.com/julienr/ipynb_playground/blob/master/keras/convmnist/keras_cnn_mnist.ipynb

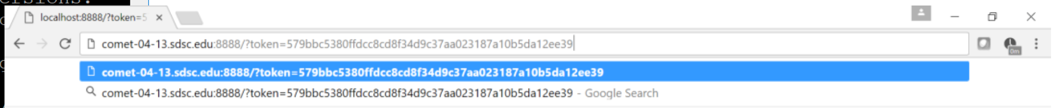
Tutorial

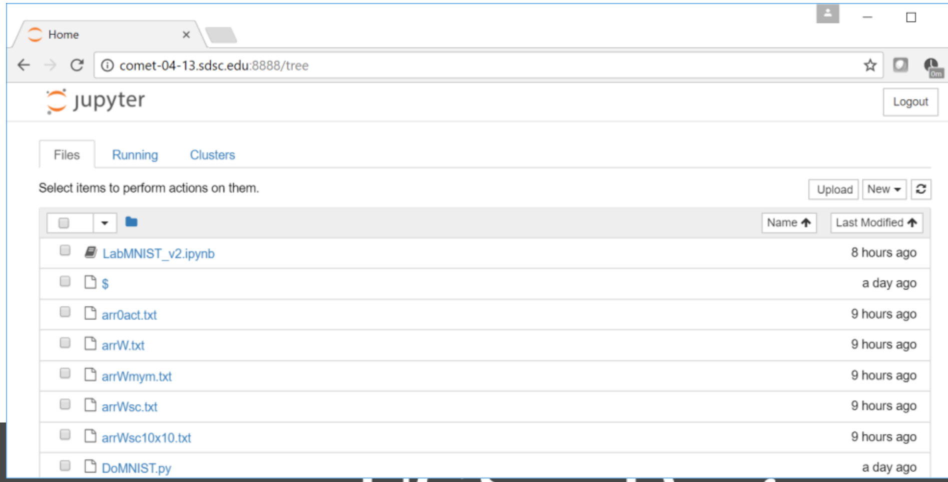
- **MNIST database of handwritten printed digits**
- **The ‘hello world’ of Conv. Neural Networks**
- **Use Keras front end (high level neural functions) to Tensorflow engine (neural math operations)**
- **Works with GPU or CPUs**

Instructions

1. Login to comet
2. Access compute node: sh Get_Inter_CompNode.sh
3. Enter: module load singularity
4. Enter: singularity shell /share/apps/gpu/singularity/sdsc_ubuntu_tf1.1_keras.img
5. Enter: ipython notebook --no-browser --ip="*" &
6. on local machine, in browser url edit box, enter the http string shown, but replace localhost with comet-XX-XX.sdsc.edu
7. Open LabMNIST_v2.ipynb
8. Run lab, review performance, view plots; change 1st convolution to 9x9 and compare







LabMNIST_Final Last Checkpoint: 7 minutes ago (autosaved)

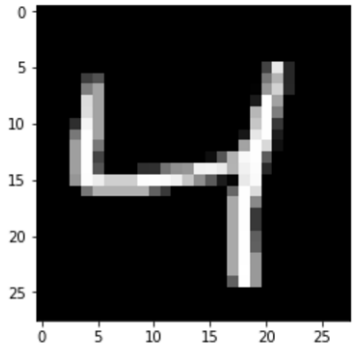
```python
for i in range(0,3):
    im = Image.fromarray(X_train[i,:,:])
    im.save("Xtrain_num"+str(i)+"_cat_"+str(Y_train[i])+".jpeg")

plt.figure()
plt.imshow(im,'gray')
plt.show()

print('img load done')
print (time.strftime("%H:%M:%S"))
```

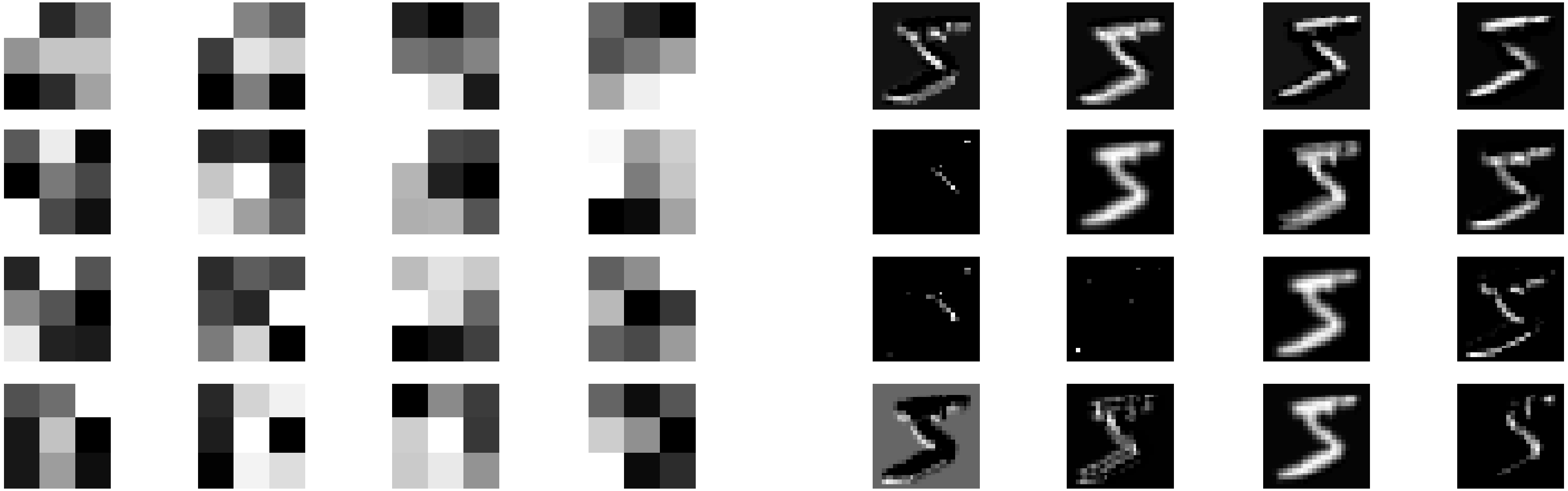
```
(5000, 28, 28)

<matplotlib.figure.Figure at 0x2ad45d04f2e8>
```

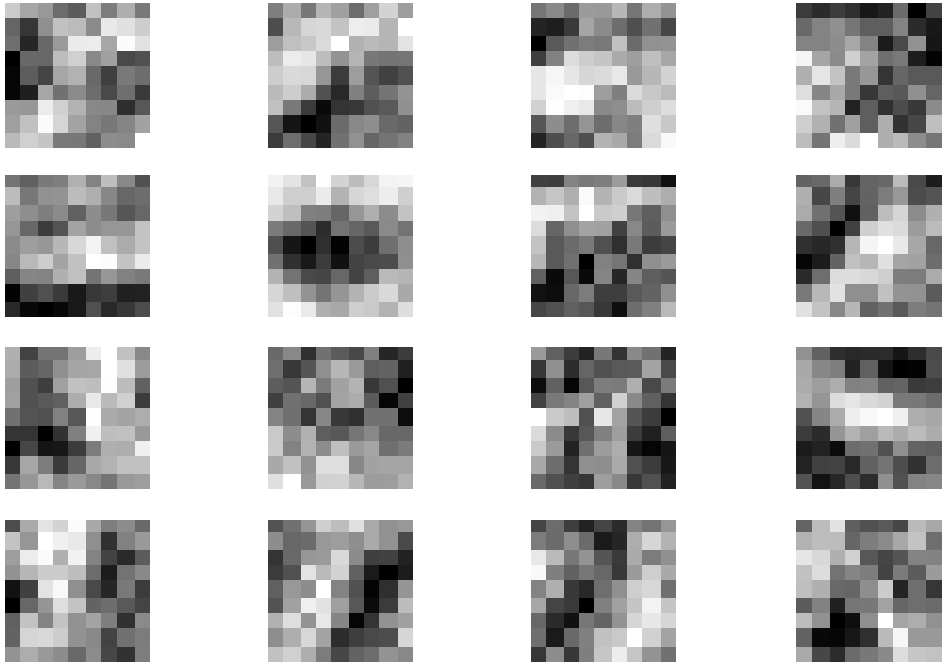


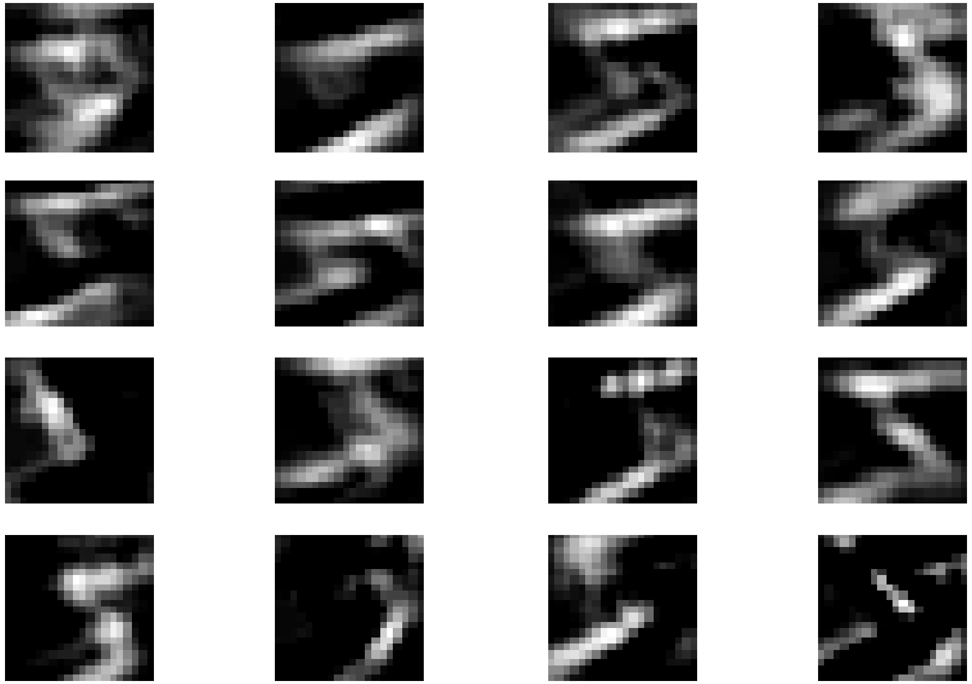

```
img load done
```

3x3 first convolution layer filter and activation



9x9 first convolution layer filter and activation





Pause

Finally

Many deep learning tools and frameworks
Other applications include time series and NLP

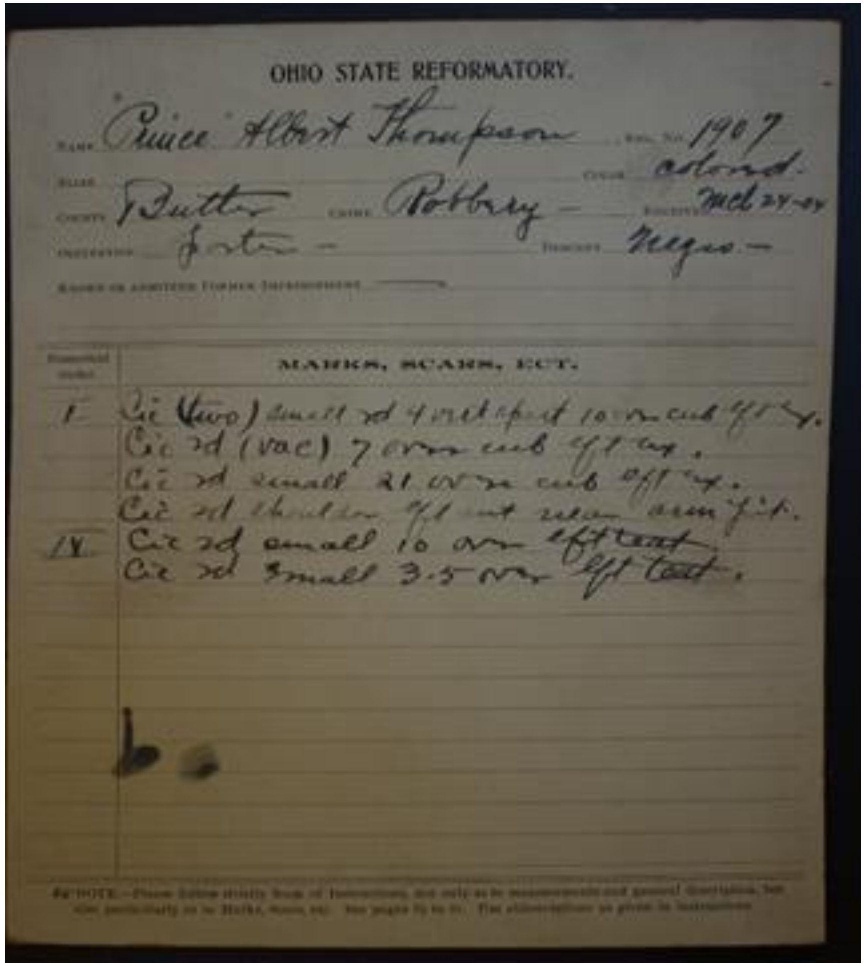
Case Study:
Character Recognition

Classifying Handwritten Cursive Text into Word Categories

Overview

- **Preprocessing**
  - RAW to PNG format conversion
  - Image dewarping
  - Brightness-contrast correction
- **Segmentation**
  - Top region extraction
  - Sub region near expect fields
  - Rotation correction
  - Word spot field name, ie ‘DESCENT’, and extract adjacent cell image
- **Word classification**

Apply standard computer vision shape and object functions (openCV)



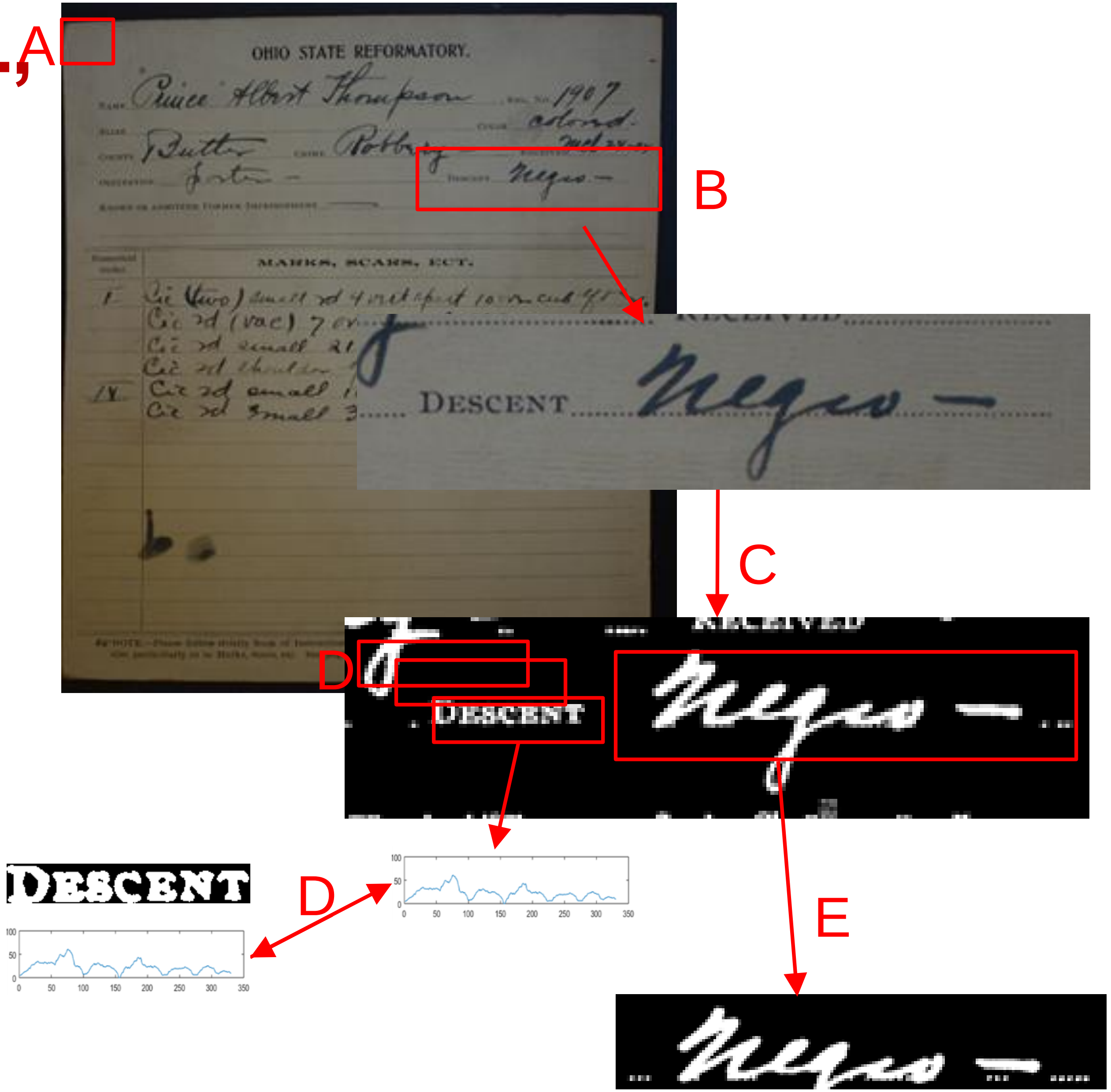
Sample Bertillon Card (back side)

Bertillon Card documenting Albert Thompson, Ohio State

Word Spotting ‘DESCENT’

A. Get upper left corner of card
B. Hard code general field region
C. Binarize
D. Search boxes to match profile of pixel concentration with a ‘DESCENT’ template
E. Extract cell of field value

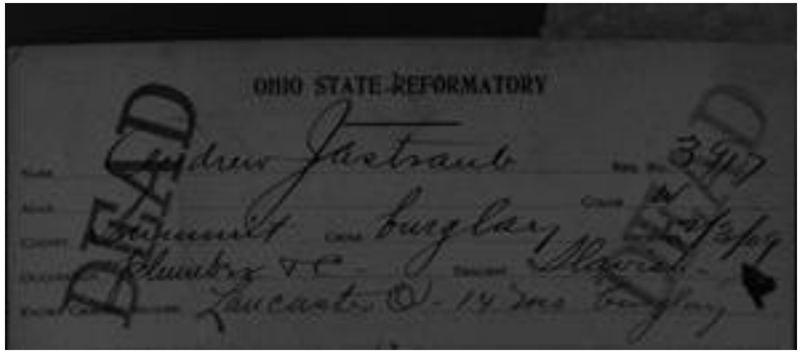
About 16-20 hours/1000 cards on Bridges HPC



Essentially, template matching with search

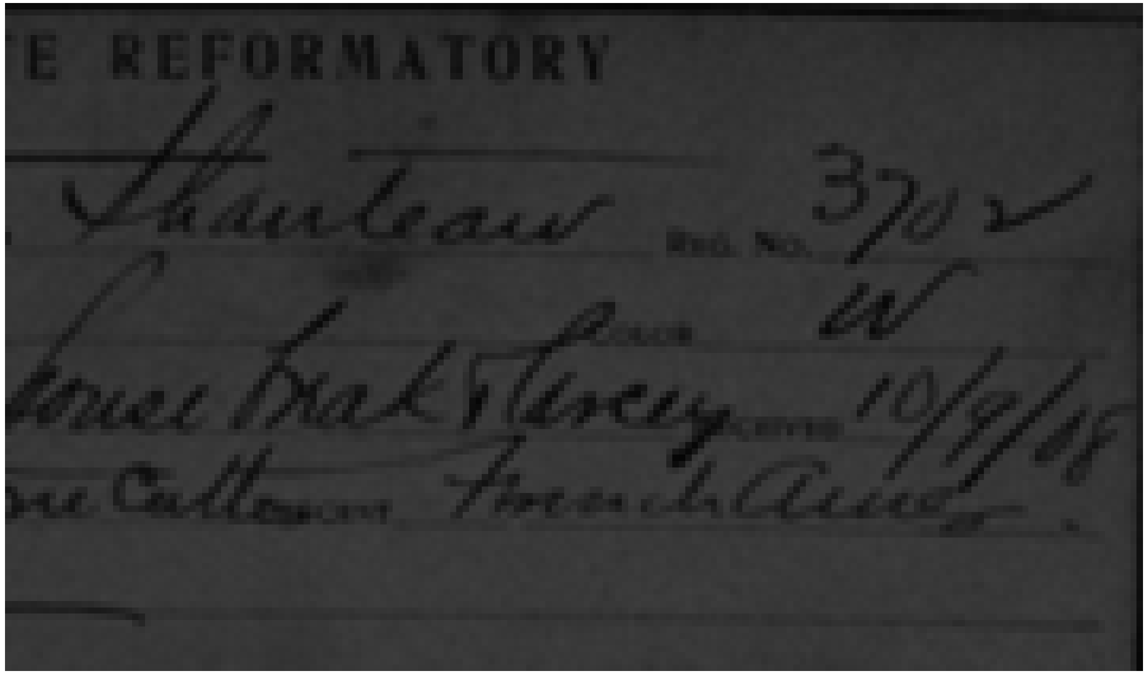
Regional Hard Spots

- Marks, tears, and blotches can throw off binarization thresholds and field localization



Field Value Extraction Issues

- Writing outside cells, over field words, intermingling letters from different cells, intermingling with dotted lines



Cursive Handwriting Recognition

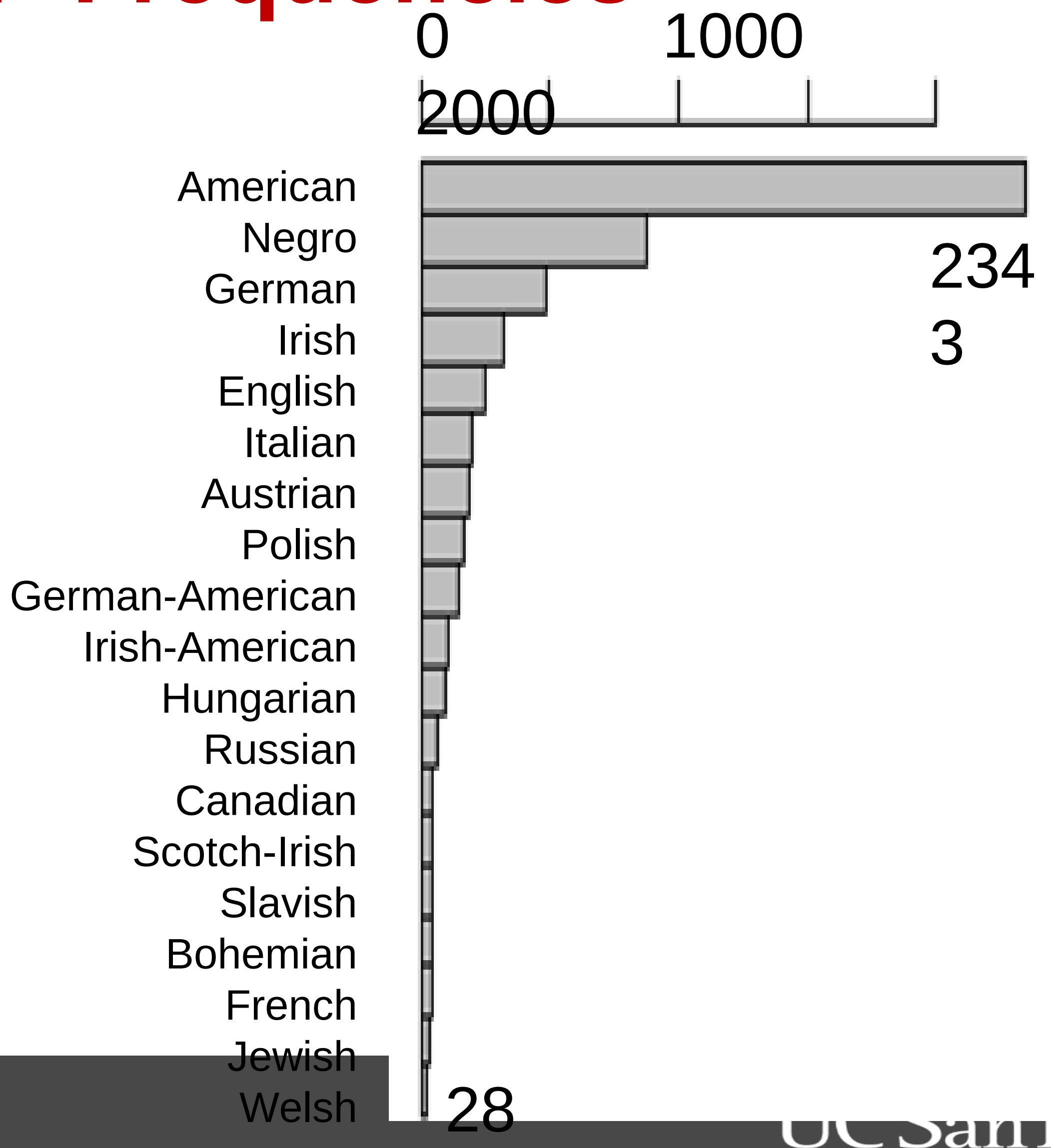
- **No standard pretrained models**
- **Best models achieve ~75-90% on large, clean, sample test sets**
  - Often with printed text
- **Goal: Explore Possibilities for Cursive OCR**

Strategy:

  - Do word classification NOT transcription (ie don't try to separate letters)
  - Limit to DESCENT and CRIME field most common values
  - Build tool for helping correct predictions

‘DESCENT’ Frequencies

- **Final Tally out of 6316 cards:**
  37% American
  175 different labels (many combinations)
  Many more different unique transcriptions



Handwriting Noise

handwriting styles, abbreviations, hanging letters, thick ink lines, dotted line, feature-less small letters, flourishes, marks, etc...

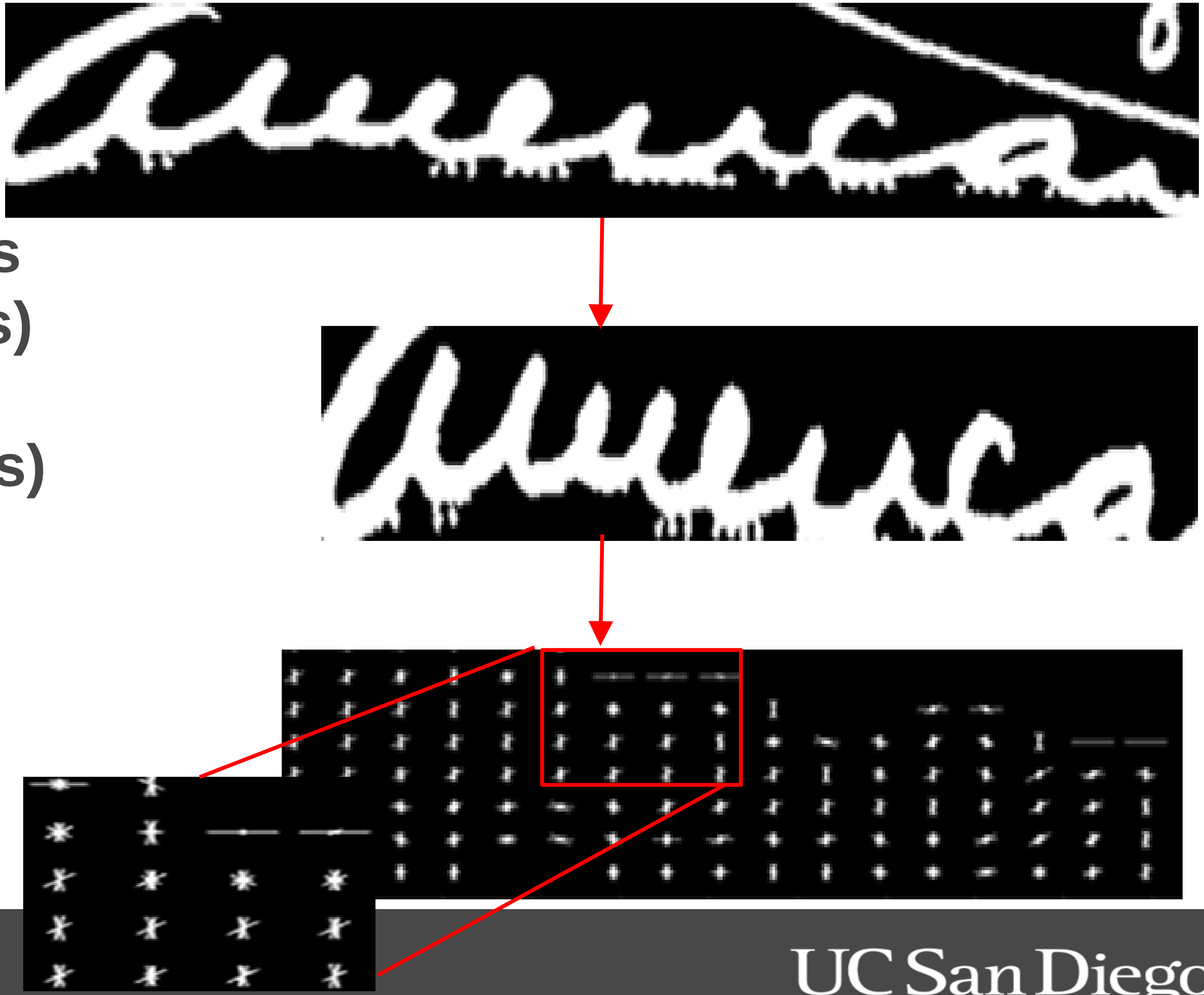




Final Denoising Heuristics

- Remove hanging marks
  (if not connected)
- Remove horizontal line of dots
  (but longer than typical letters)
- Remove lone dots
  (but smaller than letter dots)
- Resize to standard

- Get histogram of gradients within 12x12 cell as features
- Use SVM classification



Findings

‘DESCENT’ Field Word Recognition Findings

1. 343 training exemplars (manually cleaned) in 17 categories
2. First few years of Ohio State Reformatory (OSR) cards: 60-70% correct
3. Final set of 6316 total OSR cards:

    SVM Results:

    (702 not in training set, 383 poorly segmented or not readable)

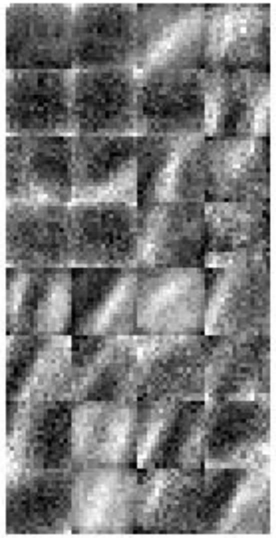
    56% (2731) correct out of remaining 4888 predictable cases

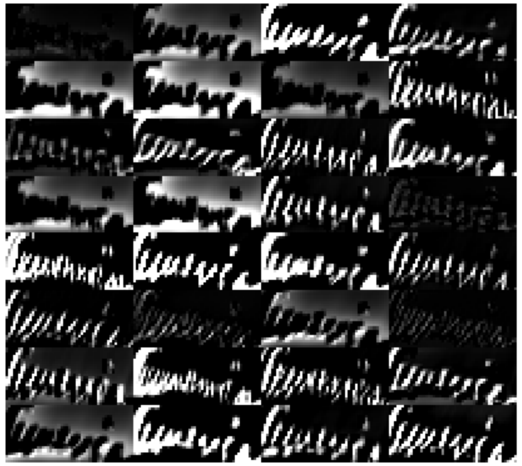
Conv. Neural Network (Deep Learning) on 'CRIME' Field

1. CNN beats SVM on 'DESCENT' when there are >1000 training exemplars
2. But CNN requires more setup and parameter searching
3. Test:
   Transfer (convolution layers) learning from 'DESCENT' to 'CRIME' field

   Results::
   Using 1095 training exemplars (not manually cleaned) in 19 categories
   830 cases not in training, 109 poorly segmented or not readable
   ~46% (1938) correct out of 4212 predictable cases
   (but depends on how you judge 'not readable')

1st Conv Layer Filters (best-activating input)

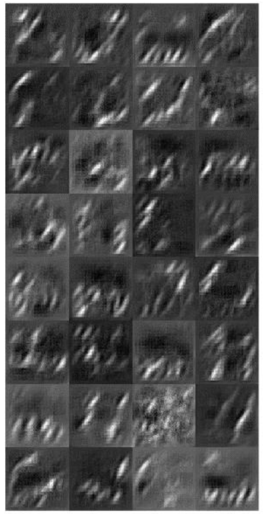




1st Conv Layer Activation for inputs of size 100x210

Output size: 16x16 filter, stride 4 =>
32 maps of 22x49

2nd Conv Layer Filters (best-activating input)



2nd Conv Layer Activation for inputs

Output size: 16x16 filter, stride 4 =>
32 maps of 3x17

Testing Amazon Mechanical Turk

- Task: CRIME and DESCENT transcription
- Results on test of 57 cards - 10 cents/card offered:
- Requires public domain images
  60 secs/card (i.e. ~$6 /hour)
  Workers started within 5 minutes of publishing task
  Workers used top half of card (ie field segmentation implied)
  Transcription mostly correct (~95%), but still requires some resolution

Summary of Considerations for hard OCR

- **Review transcription vs categorization preferences**
- **Make or get a tool to help transcribe/categorize**
- **Depending on images, task, and noise:**
    - <1000 images, get an undergrad
    - 1000-10000 images, an undergrad and maybe simple OCR
    - > 10000 images, try CNN OCR
- **Consider only using strongest predictions from a model**
- **Consider Mech. Turks if public domain data and task is easy**