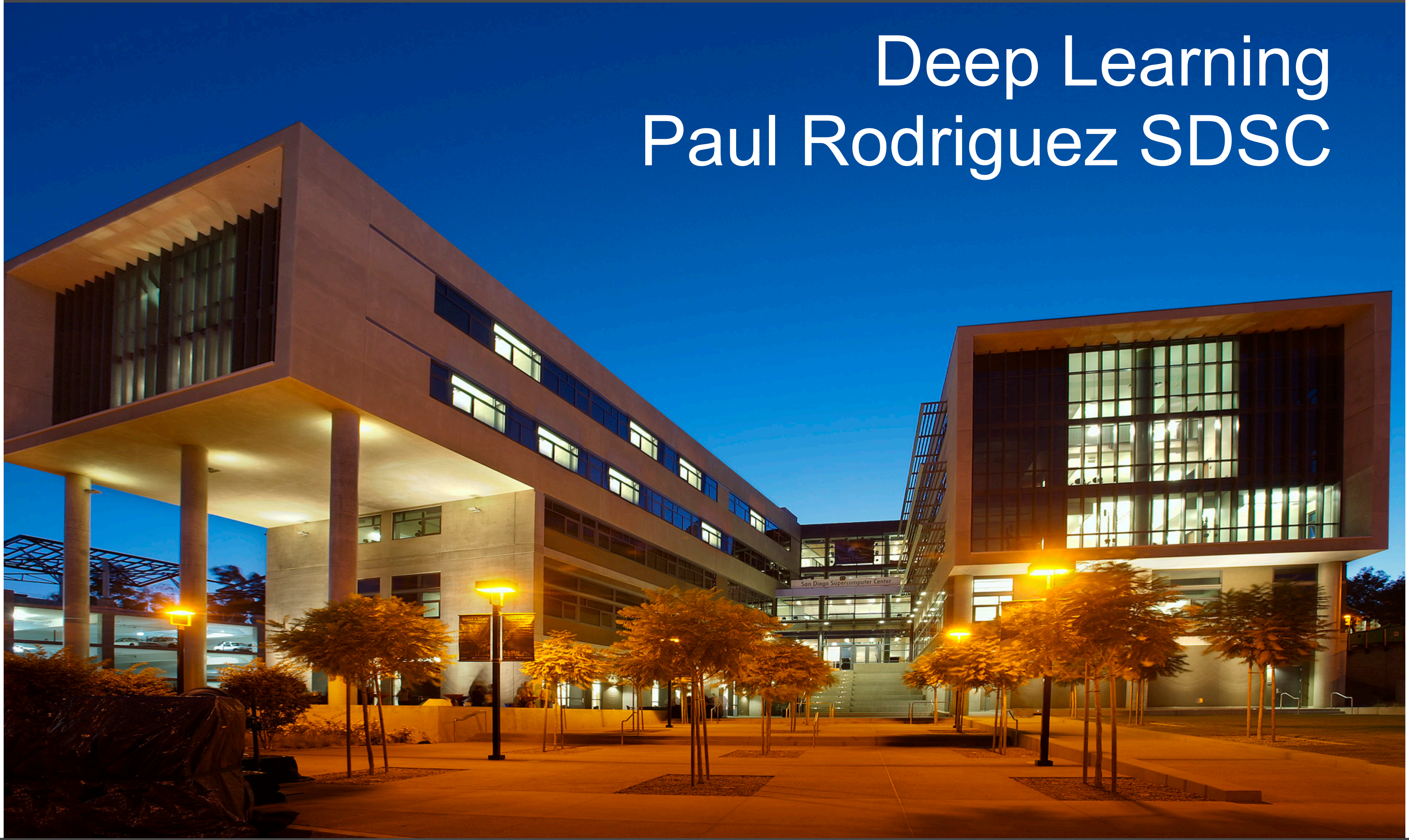
Deep Learning
Paul Rodriguez SDSC



Outline

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

Deep Learning

- **3 characterizations:**

    1. Learning complicated interactions about input

    2. Learning complex feature transformations

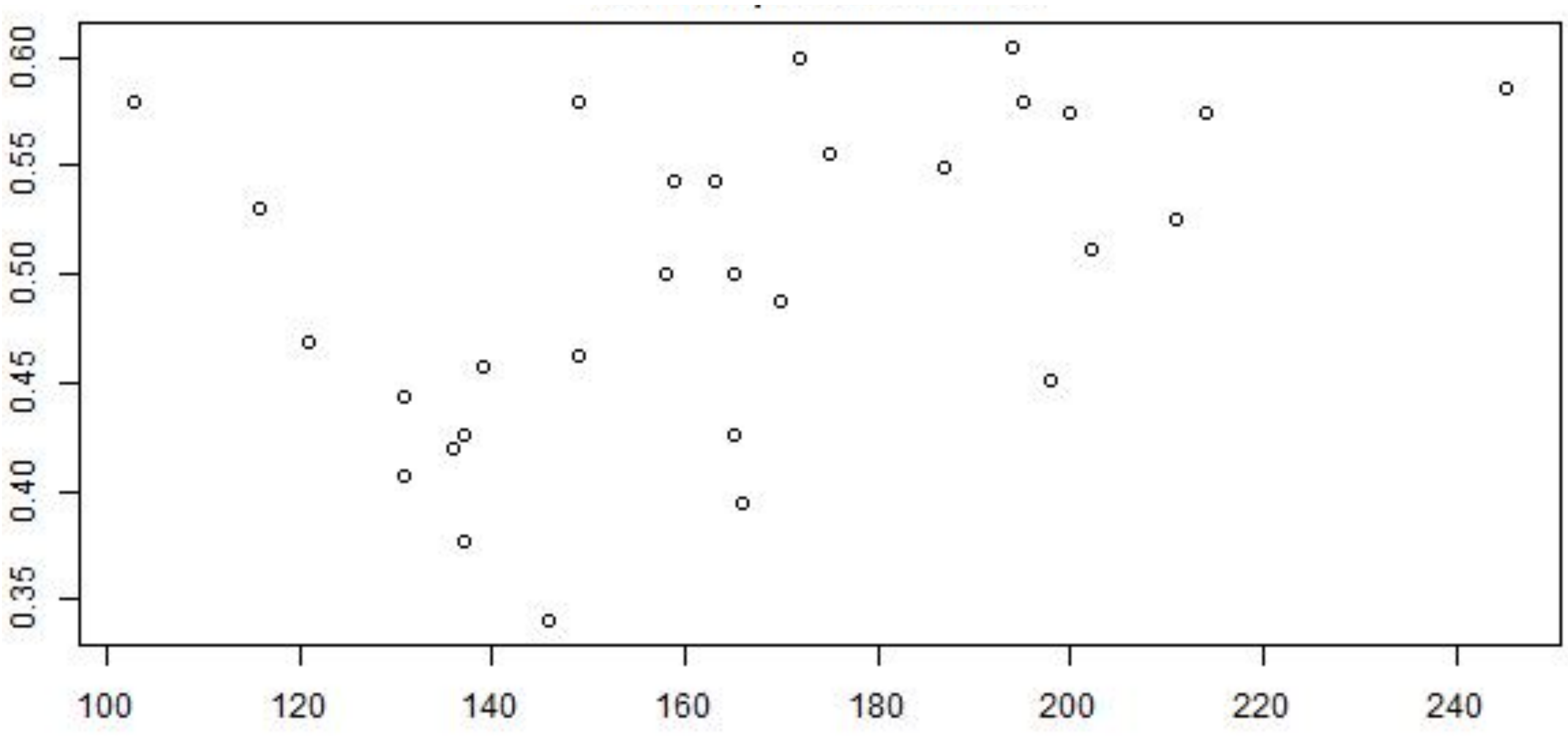
    3. Using neural networks with many layers

Explanation Strategy: Start with linear regression and go deep

A data example:Home Runs and W-L percent

W-L %

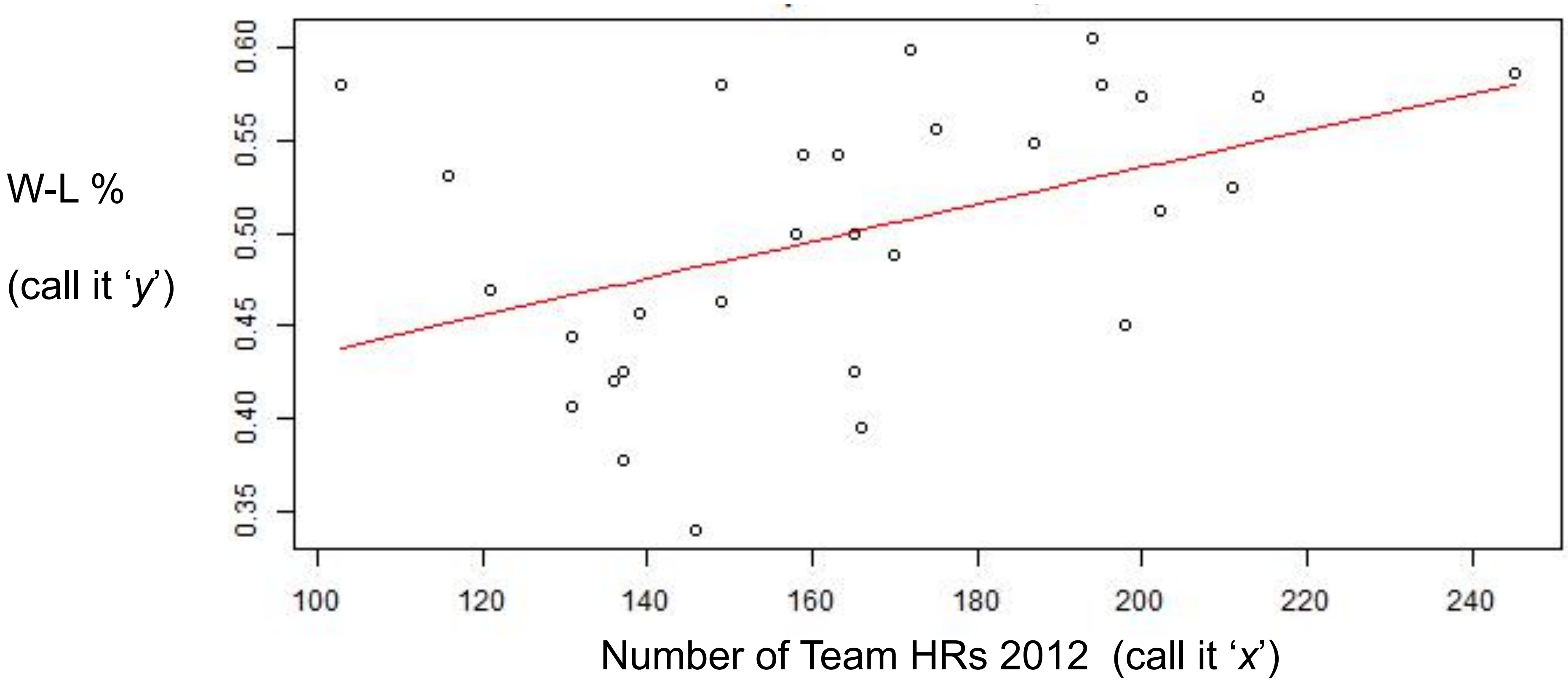
(call it ‘*y*’)



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

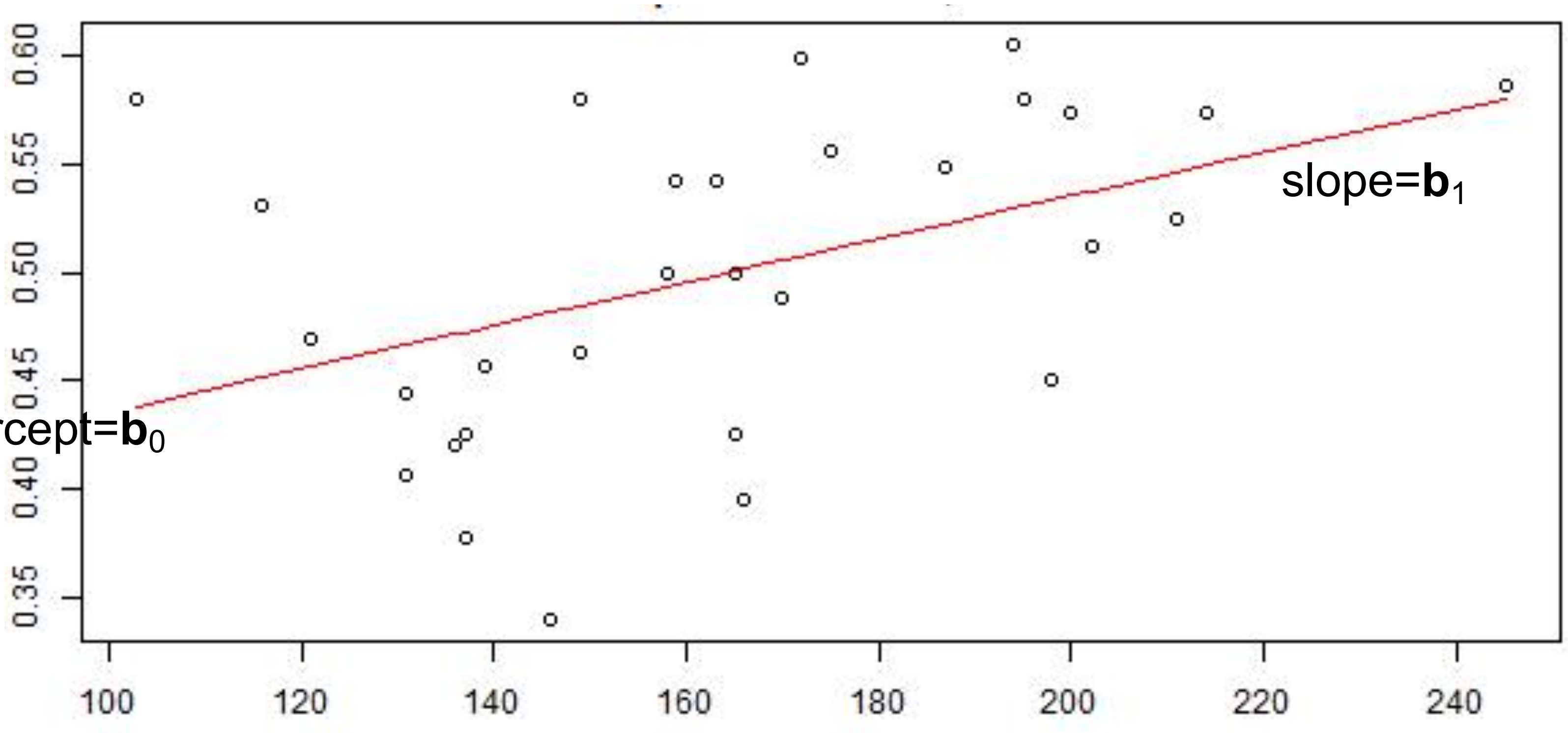
Recall Linear Regression is Fitting a Line

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



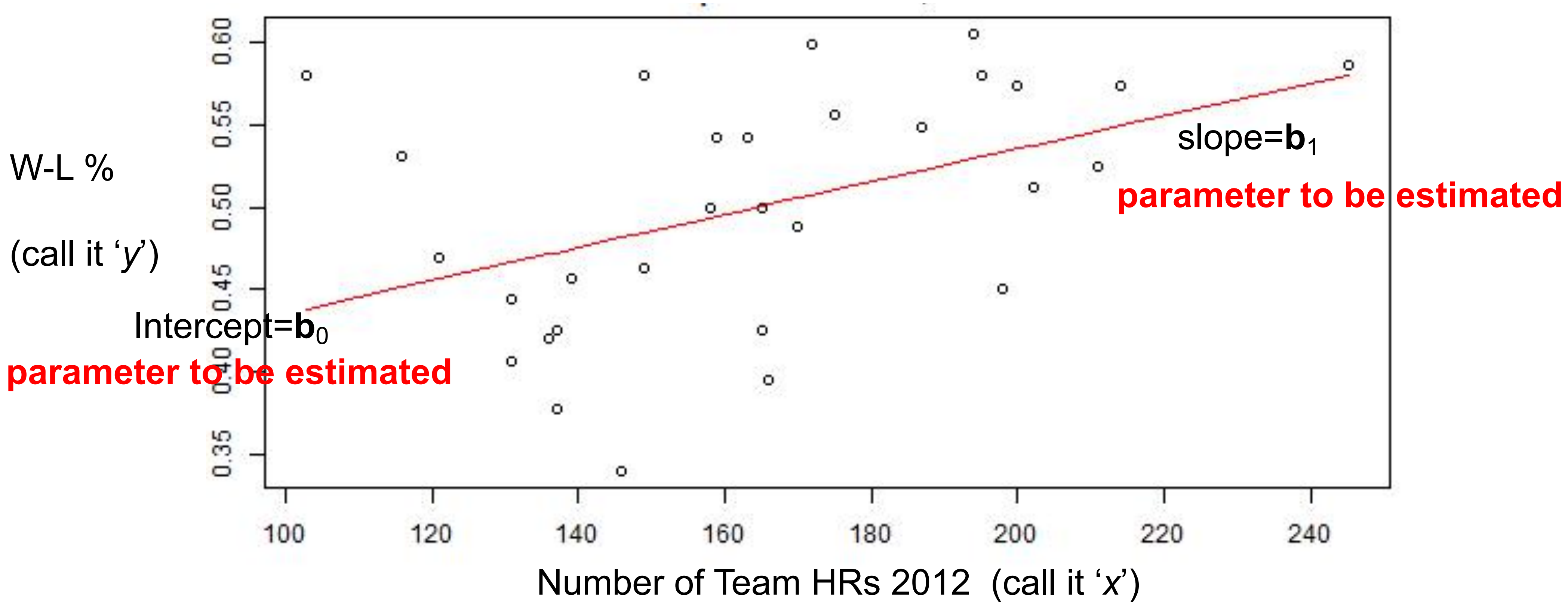
Recall Linear Regression is Fitting a Line

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



Recall Linear Regression is Fitting a Line

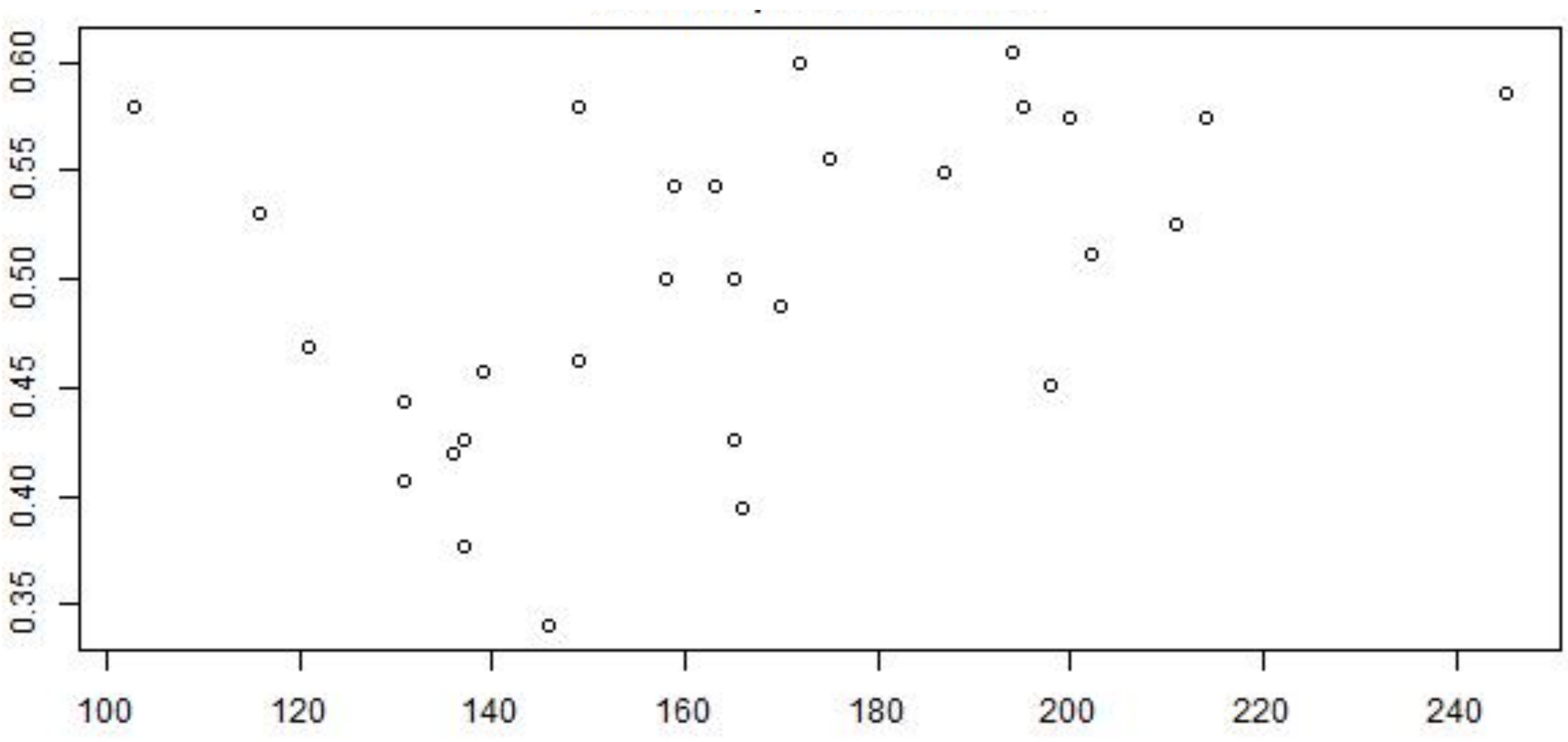
the Model: $y = f(x, b) = bo * 1 + b_1 * x$



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

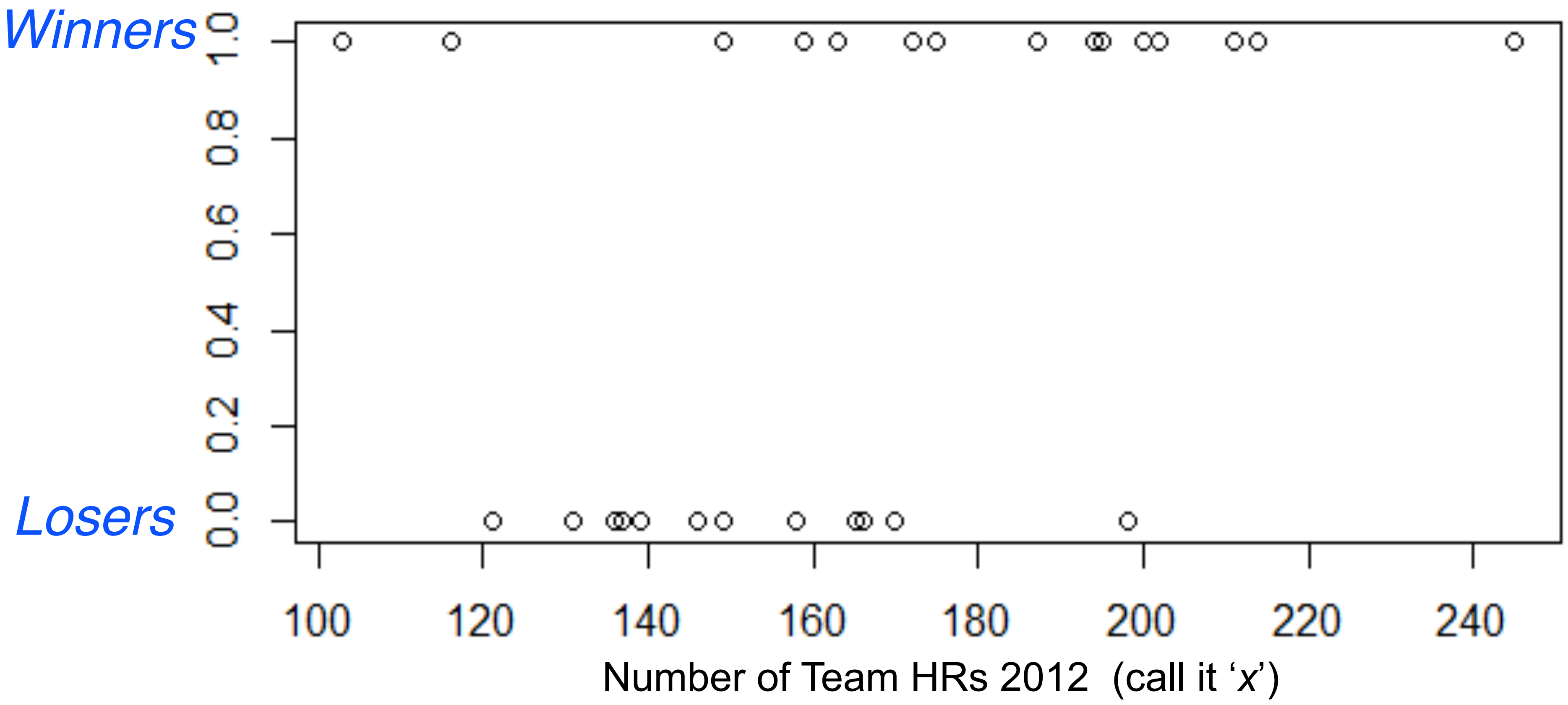
W-L %

(call it ‘*y*’)



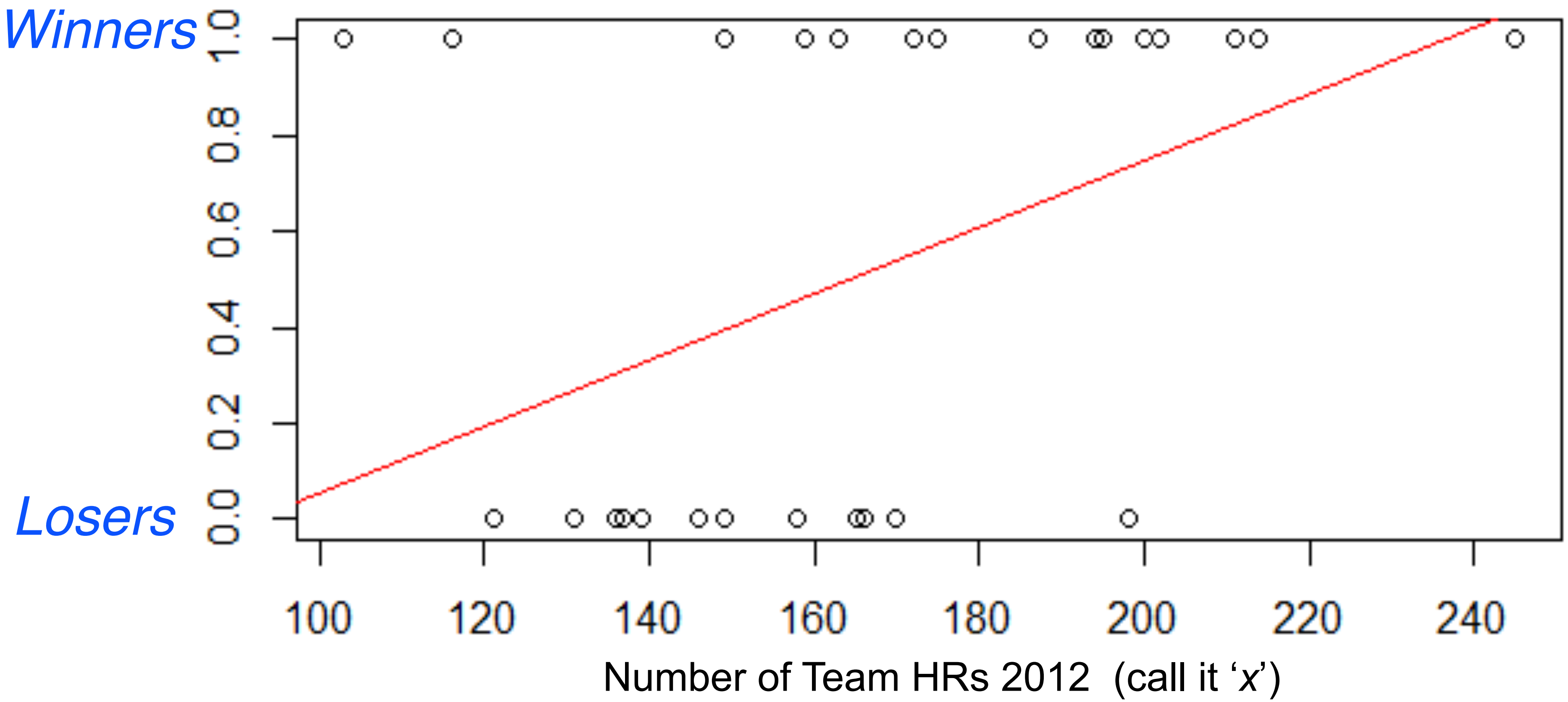
Number of Team HRs 2012 (call it ‘*x*’)

Classification uses labelled outcomes



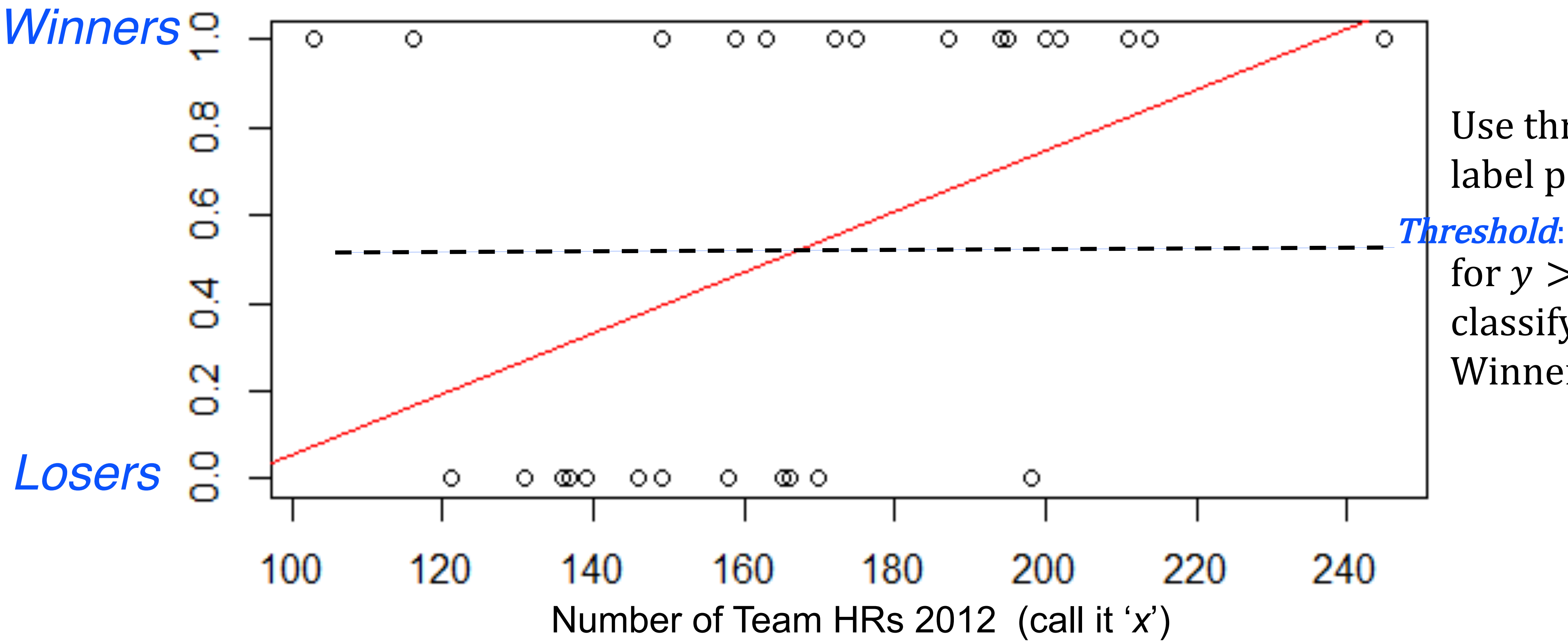
Can do this: fit a line with same model

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



Can do this: fit a line with same model

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



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: $y = f(x, b) = bo * 1 + b_1 * x$

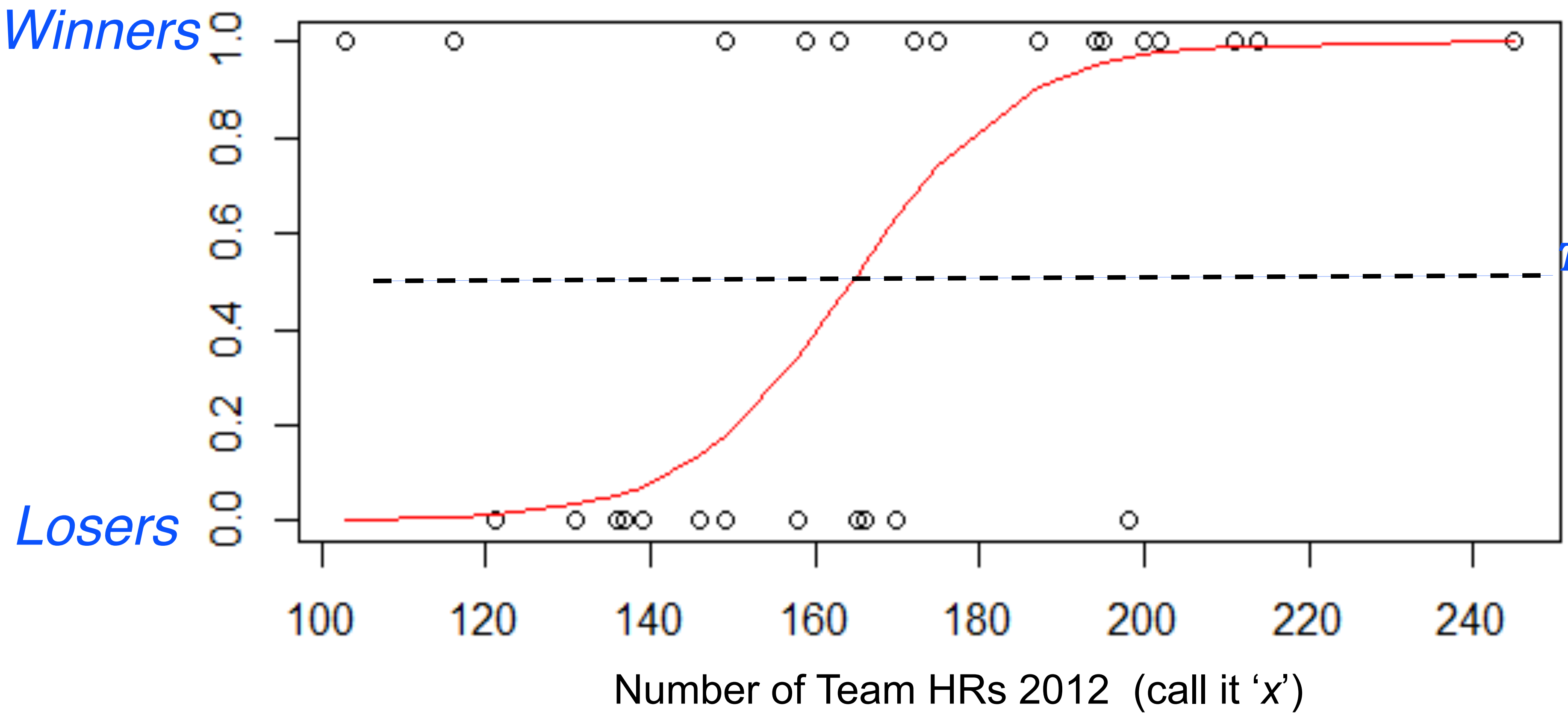
Use threshold to label prediction:

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

Divides X into Winner/Loser regions

Can do better: fit a nonlinear function

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

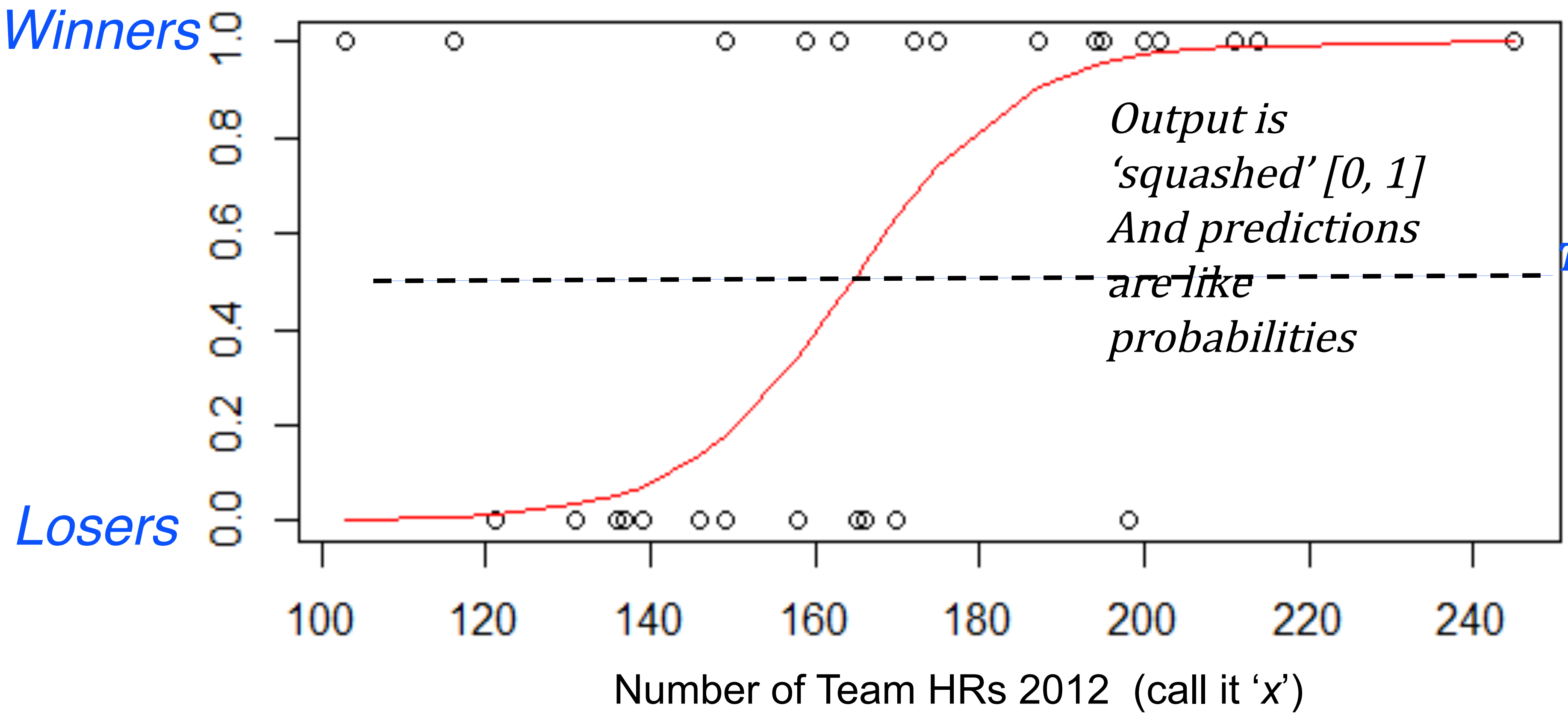


Use threshold to label prediction:

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

Can do better: fit a nonlinear function

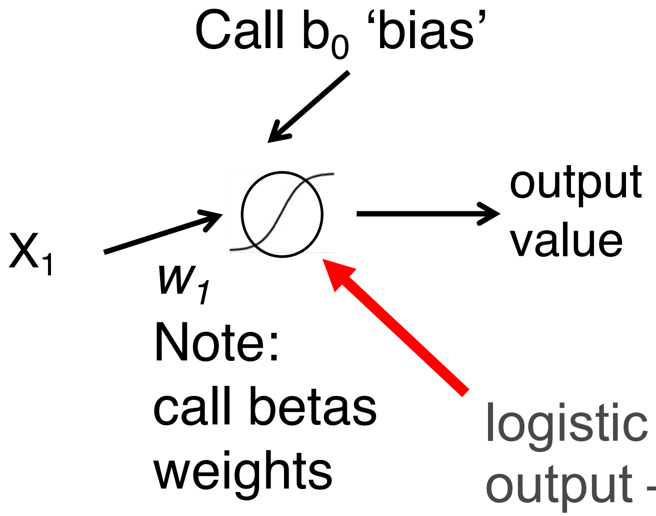
the Model: $y = f(x, b) = 1/(1 + \exp[-(b_o * 1 + b_1 * x)]$



Use threshold to label prediction:

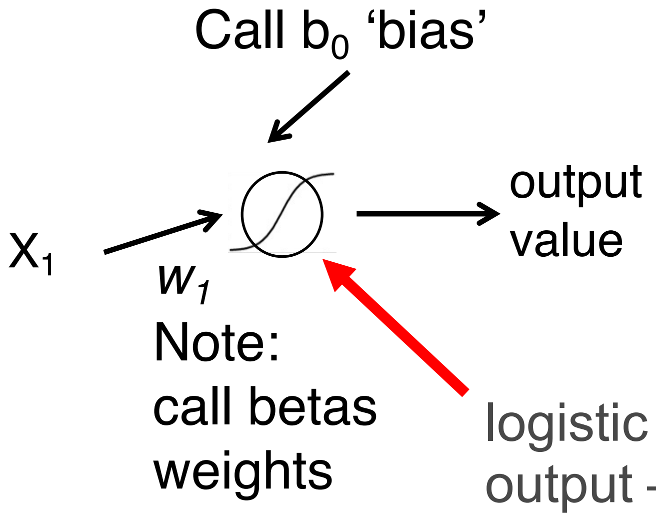
Threshold: for $y > 0.5$, classify x as Winner

Logistic Regression as 1 node network



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

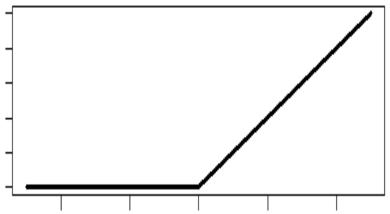
Logistic Regression as 1 node network



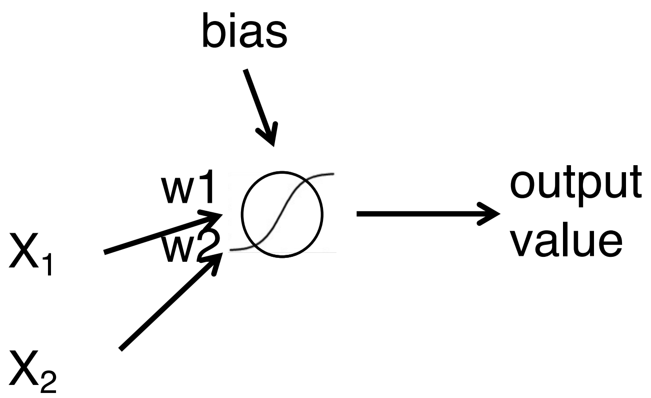
logistic function to transform input to output – call it the 'activation' function

Note: other activations are possible,

RELU (rectified linear unit)



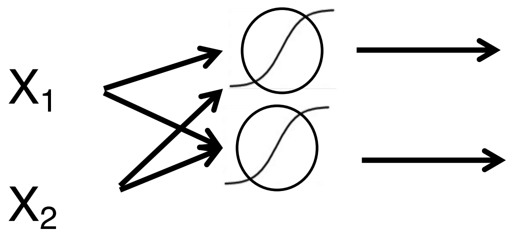
Next step: More general networks



Add input variables

More general networks

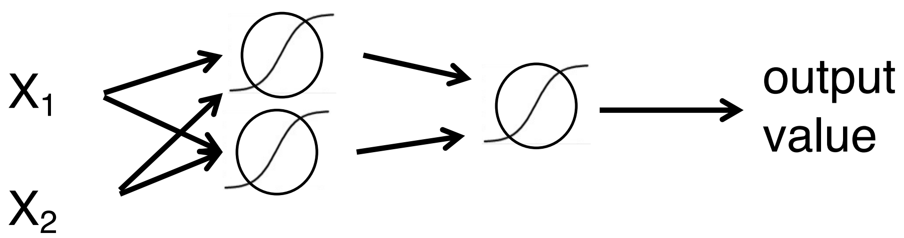
(assume bias present)



Add input variables *Add logistic transformations …*

More general networks

(assume bias)



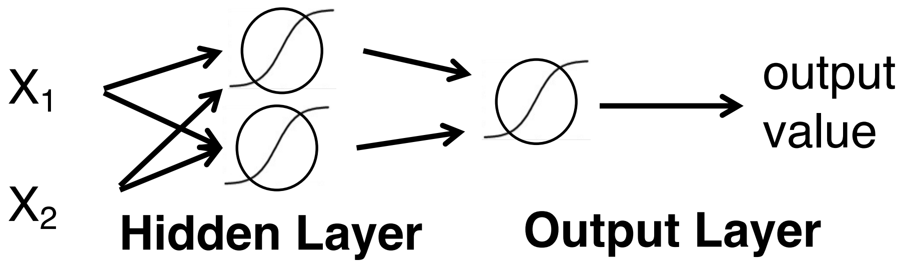
Combine transformations!

Add input variables

Add logistic transformations …

More general networks

(assume bias)



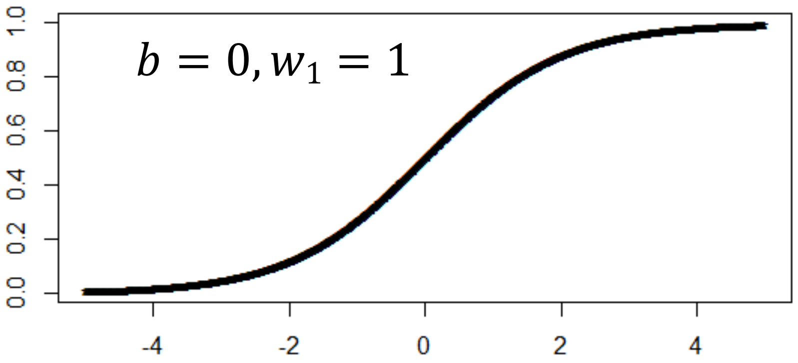
Combine transformations!

Add input variables

Add logistic transformations …

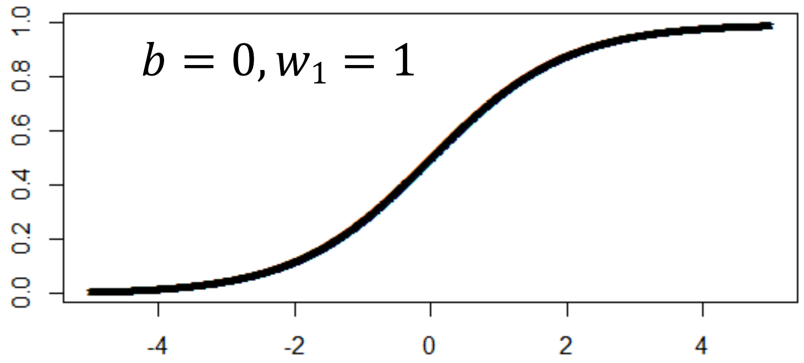
Logistic function w/various weights

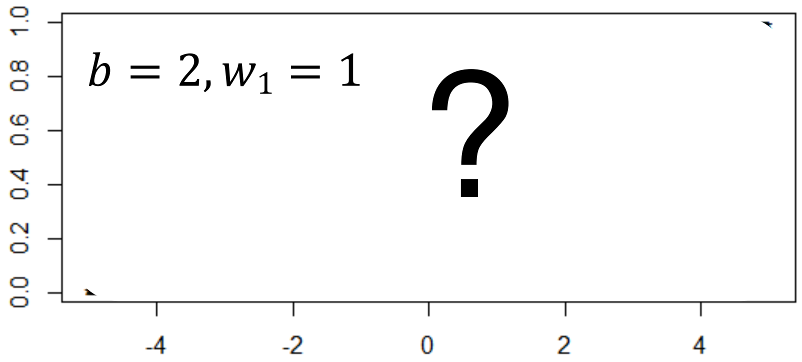
$$for\ y = 1/(1+exp(-(b+w_1*x))$$



Logistic function w/various weights

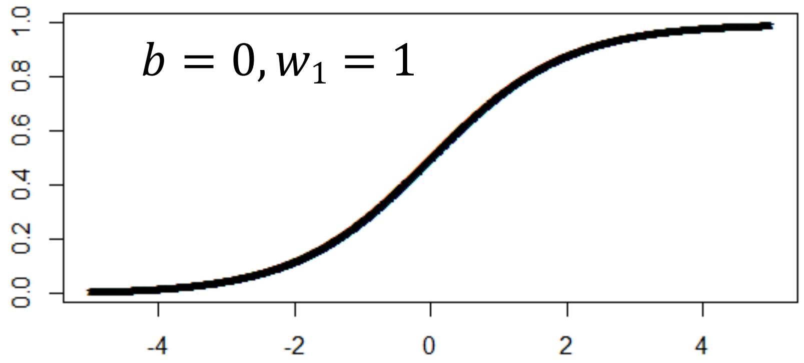
$$for\ y = 1/(1+exp(-(b+w_1*x))$$

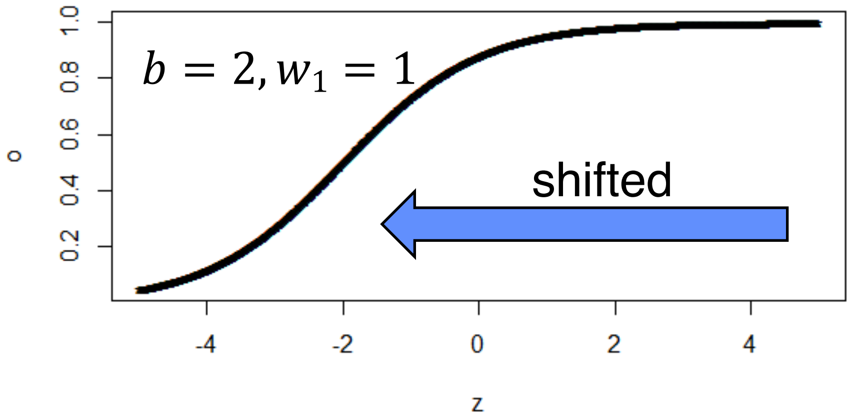




Logistic function w/various weights

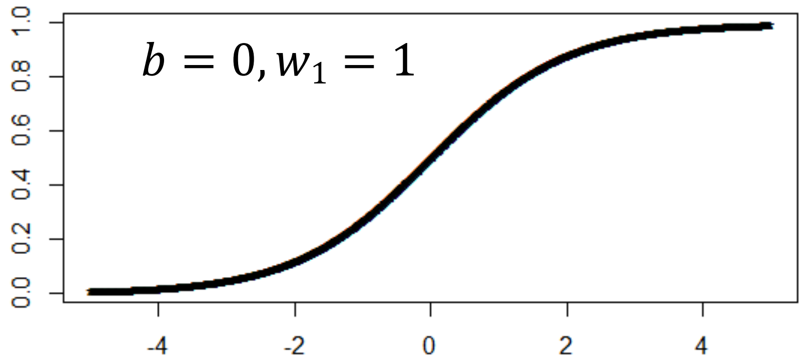
$$for\ y = 1/(1+exp(-(b+w_1*x))$$

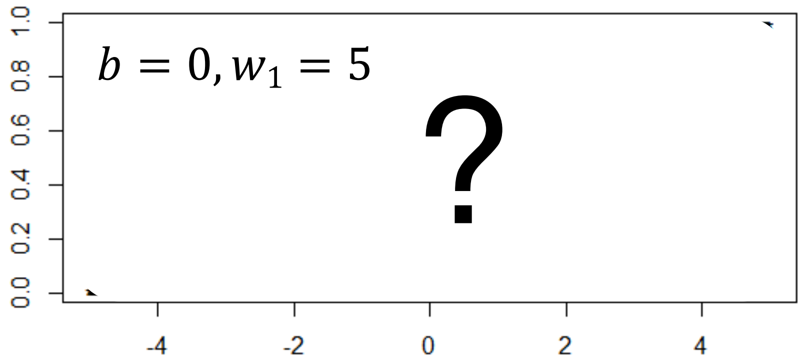


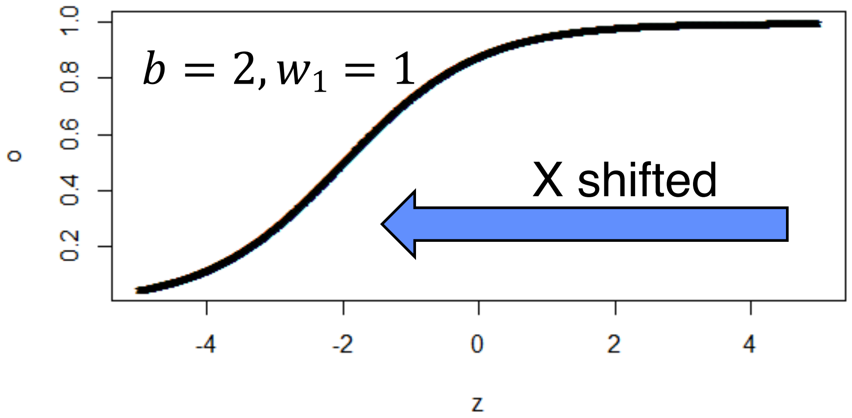


Logistic function w/various weights

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

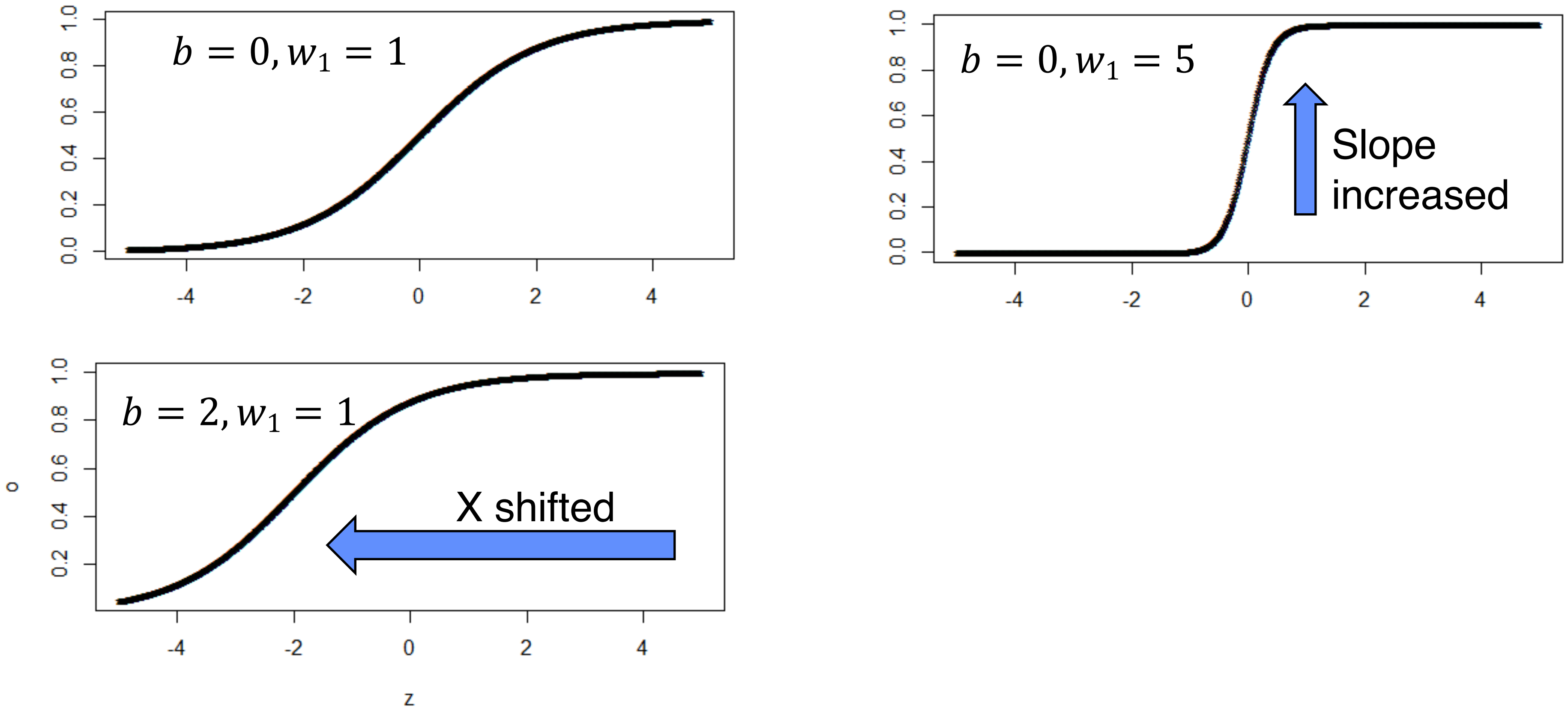






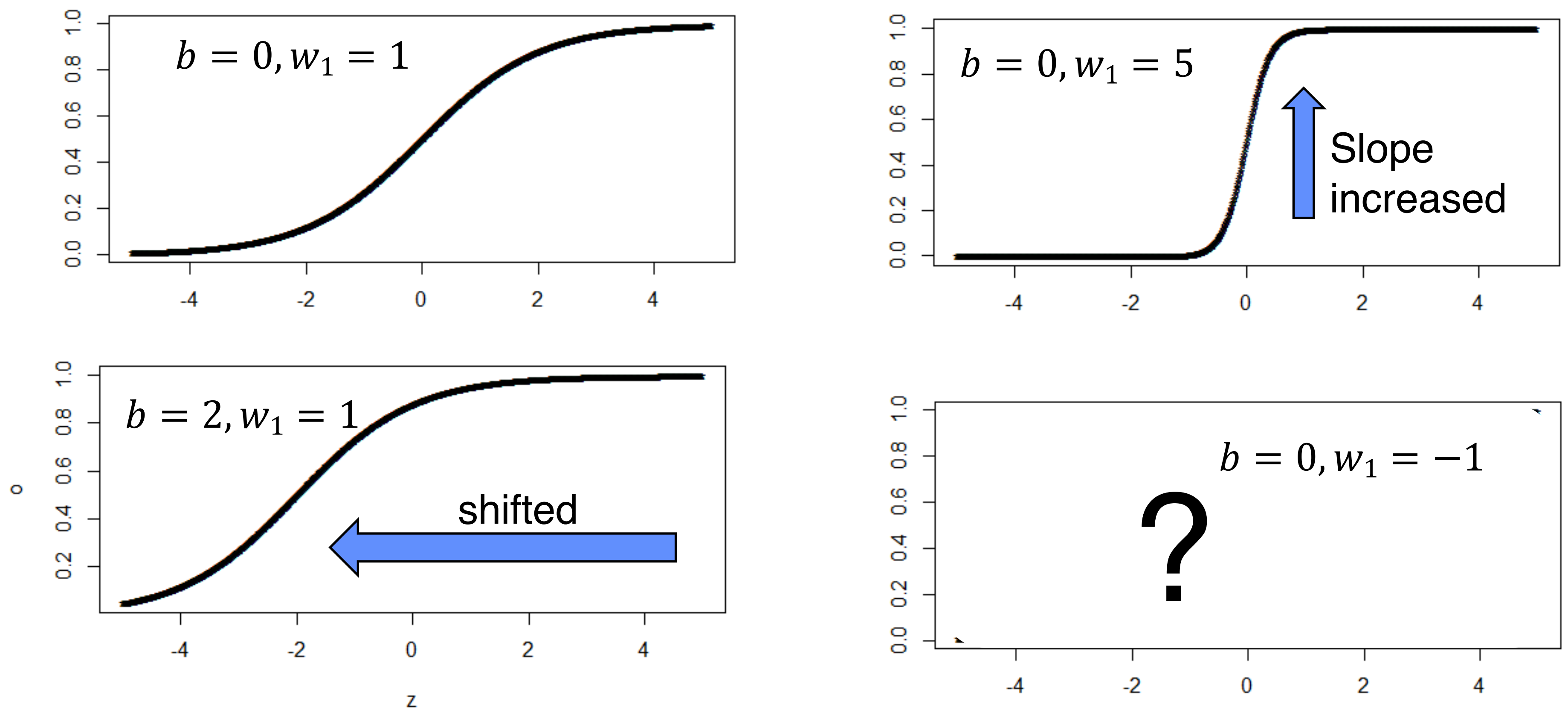
Logistic function w/various weights

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



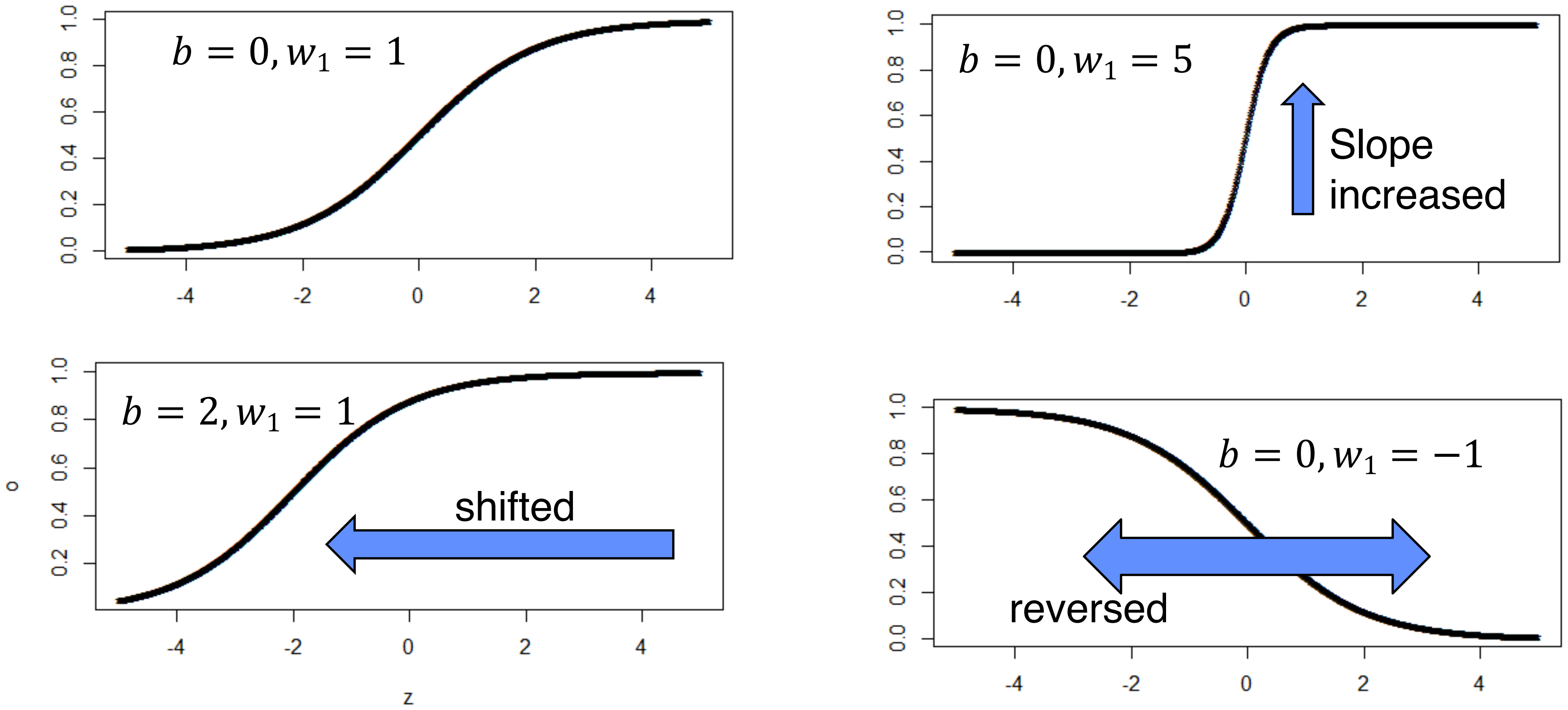
Logistic function w/various weights

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

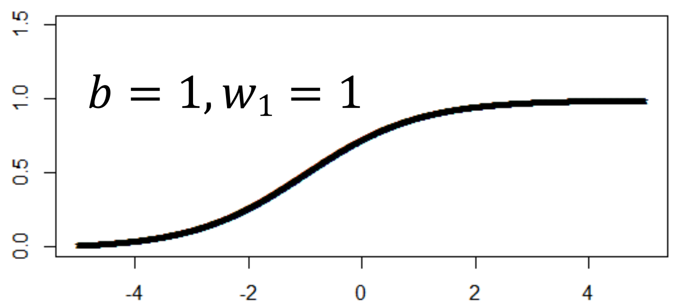


Logistic function w/various weights

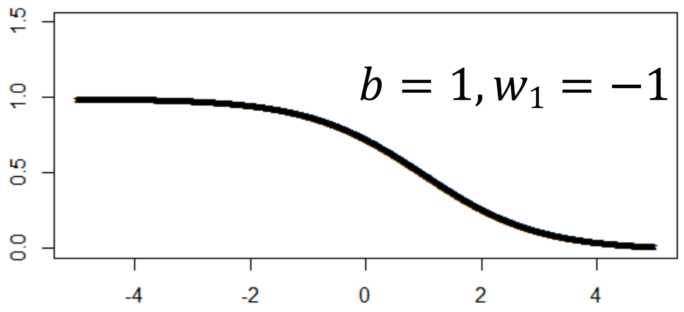
$$for\ y = 1/(1+exp(-(b+w_1*x))$$



So combinations are highly flexible and nonlinear

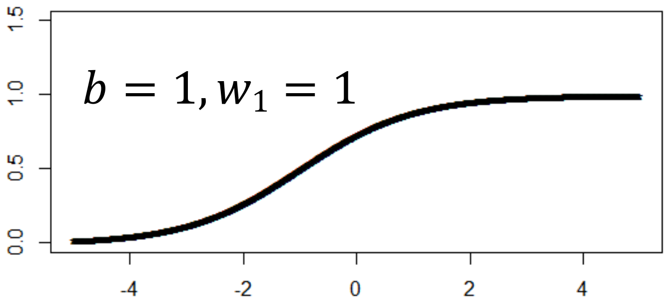


+

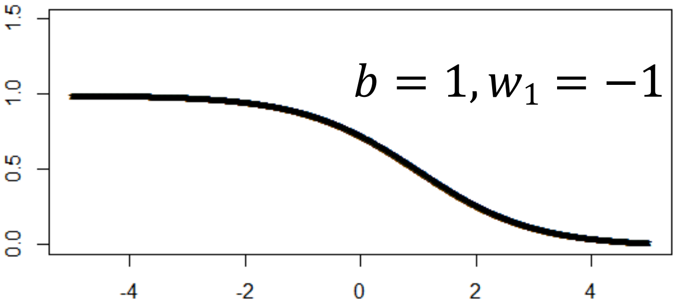


=

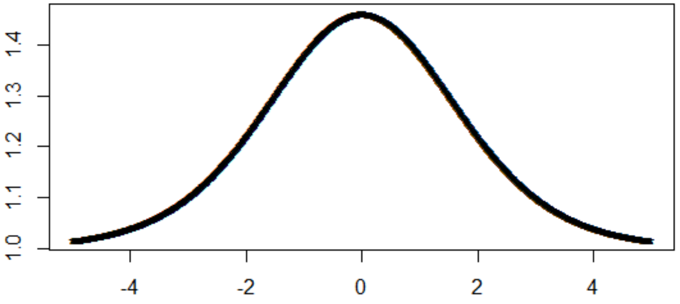
So combinations are highly flexible and nonlinear



+

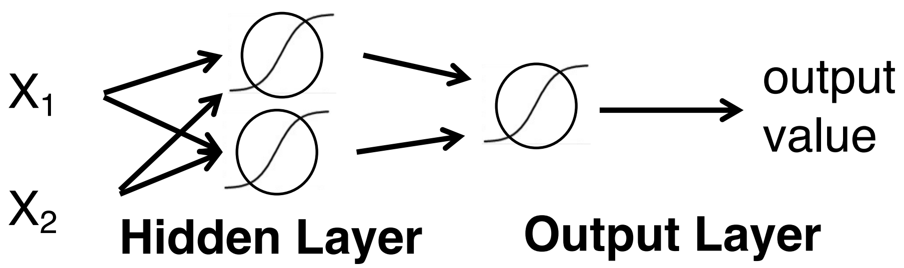


=



But parameter fitting is harder too

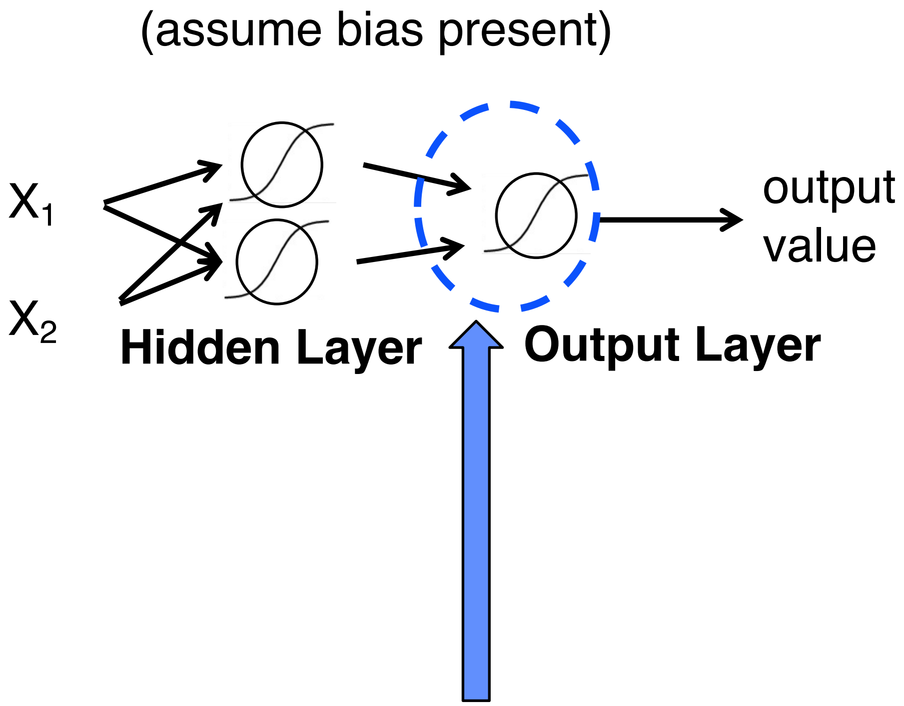
(assume bias present)



For each data instance:

Error=Output – Target

But parameter fitting is harder too

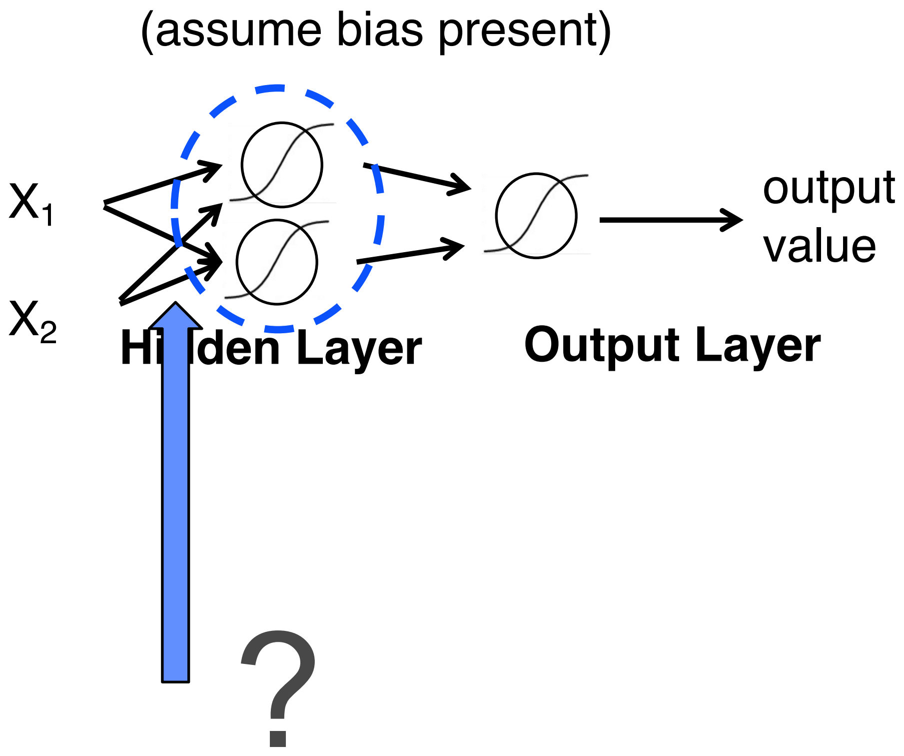


For each data instance:

Error=Output – Target

The objective is to minimize
Error related to output weights
(same as for logistic regression)

But parameter fitting is harder too



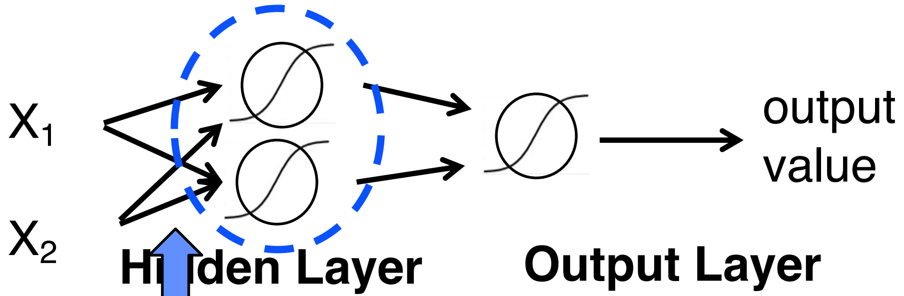
For each data instance:

Error=Output – Target

But, error signals only known for output layer, what is error for hidden layer?

But parameter fitting is harder too

(assume bias present)



For each data instance:

Error=Output – Target

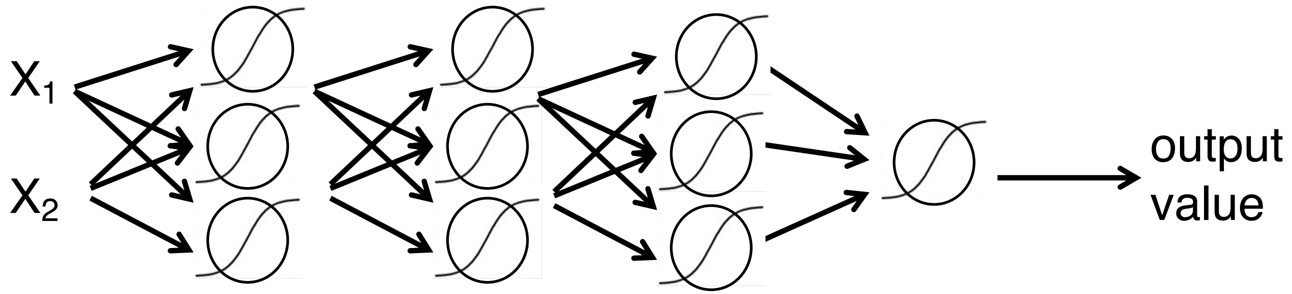
But, error signals only known for output layer, what is error for hidden layer?

Minimize *Error* related to output weights, that is also related to hidden weights

(Use derivatives to 'back-propagate' errors, "stochastic gradient descent")

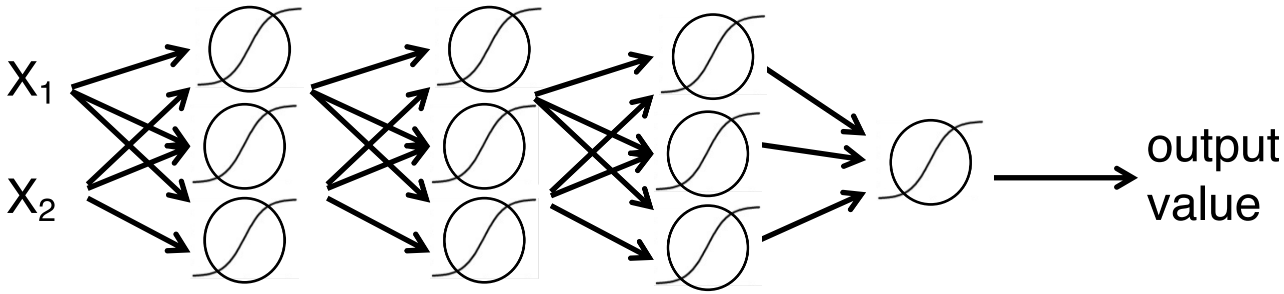
Why stop at 1 hidden layer?

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



Train with Care

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

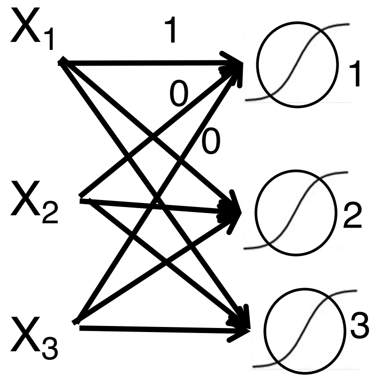


Many more parameters, and error signal at final output layer gets drowned out at lower layers-
but penalizing weight sizes, varied activation functions, and more data help!

Feature Transformations, Projections, and Convolutions

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume all b=0)



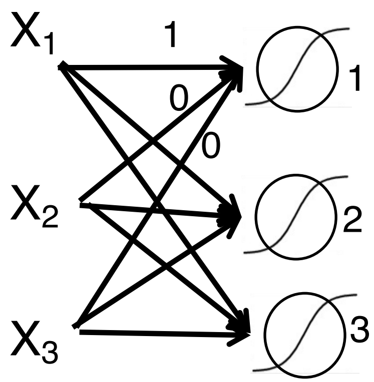
Call the connection parameters 'weights'.

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 $b=0$)



Call the connection parameters ‘weights’.

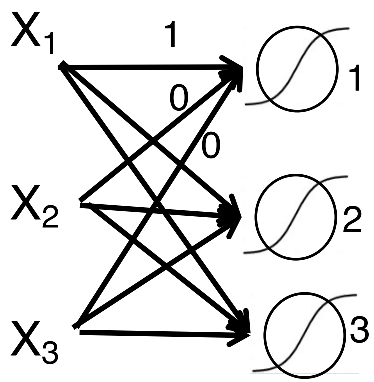
For node 1 let $[w_1\ w_2\ w_3] = [1\ 0\ 0]$

What feature transformation is that?

Informally, squash x_1 and ignore x_2 , x_3

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume $b=0$)



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

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

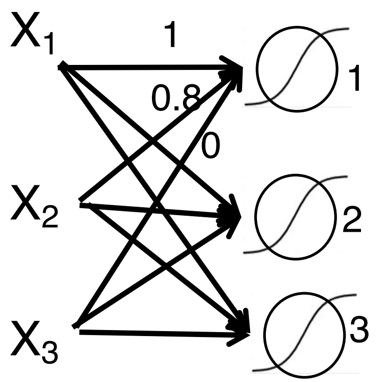
For node 3 let $[w_1\ w_2\ w_3] = [0\ 0\ 1]$

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
(assume $b=0$)



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

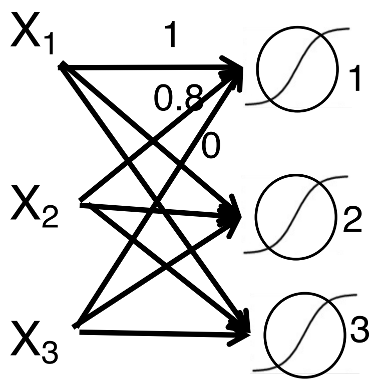
For node 2 let $[w_1\ w_2\ w_3] = [-0.8\quad 1\quad 0]$

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
(assume *b=0*)



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 $[w_1\ w_2\ w_3] = [0 \quad 0 \quad 0]$

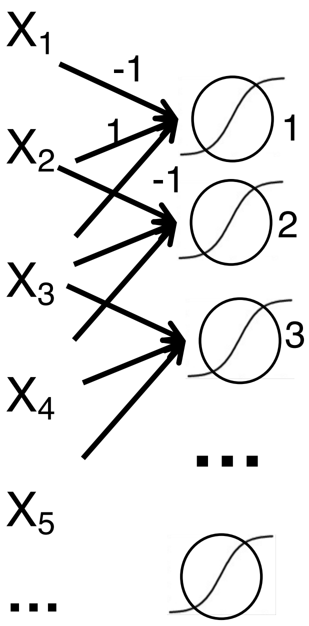
What feature transformation are these together?

Informally, like projection onto 2 orthogonal dimensions (similar to SVD)

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



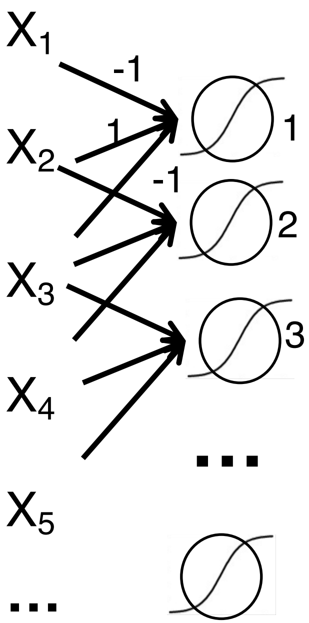
For node 1 let W= $[w_1\ w_2\ w_3] = [-1\ 1\ -1]$

What values of x_1, x_2, x_3 will give maximum node 1 output? (assuming $-1 \leq x \leq 1$)

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume *b=0*)



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

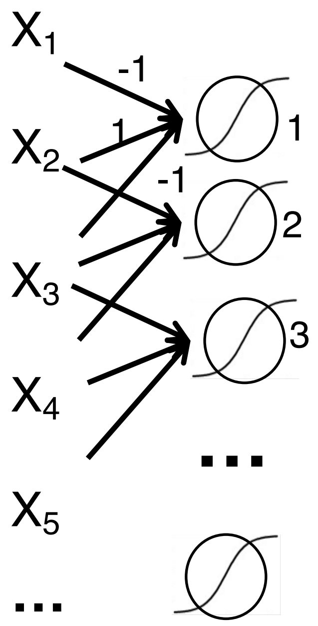
What values of x_1 ,x_2 , x_3 will give maximum node 1 output? (assuming $-1 \leq x \leq 1$)

Informally, node 1 has max activation for a 'spike', e.g. when [x_1 ,x_2 , x_3] = *[-1 +1 -1]*

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



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

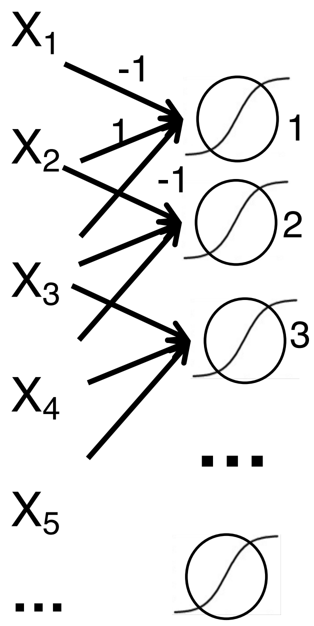
For node 2,3, etc… copy W for node 1

What is the hidden layer doing?

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume *b=0*)



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?

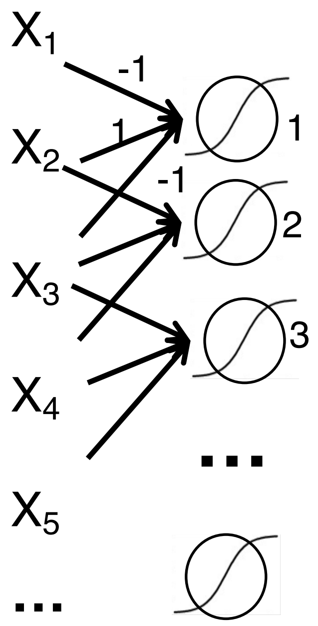
Informally, looking for a spike everywhere.

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

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



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.

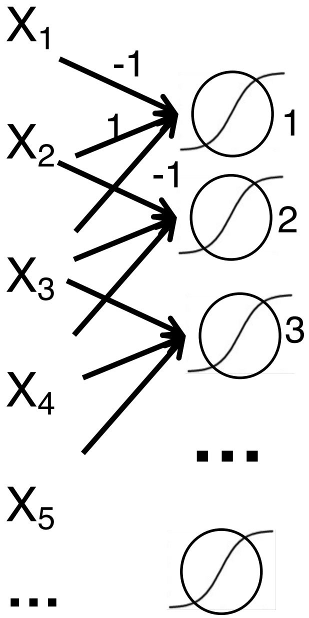
This is essentially a convolution operator, where W is the kernel.

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

A Filter

Many X input, but only 3 connections to each hidden node from the 'local' input, i.e. a receptive field

(assume $b=0$)



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.

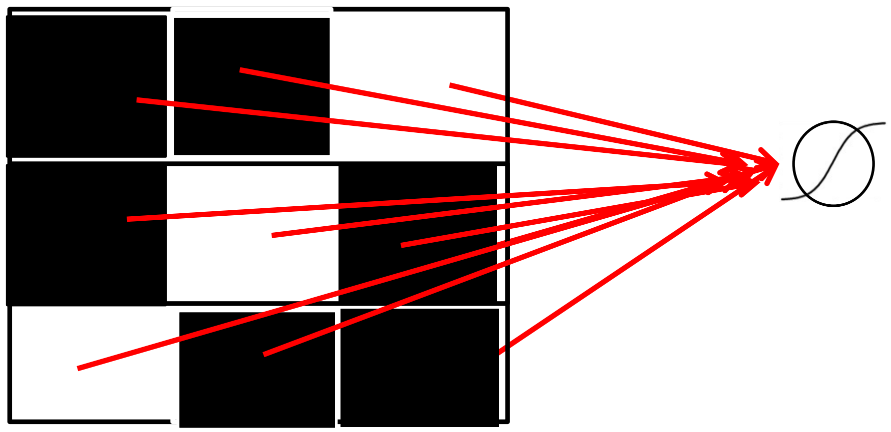
This is essentially a convolution operator, where W is the kernel.

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

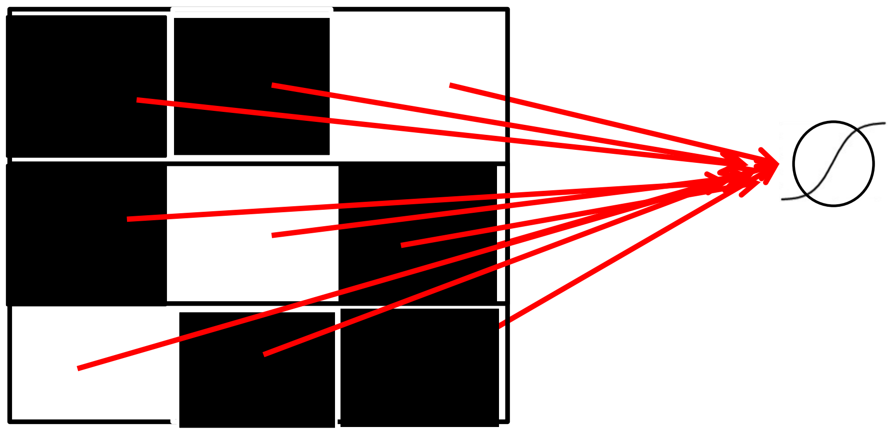
Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node



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)

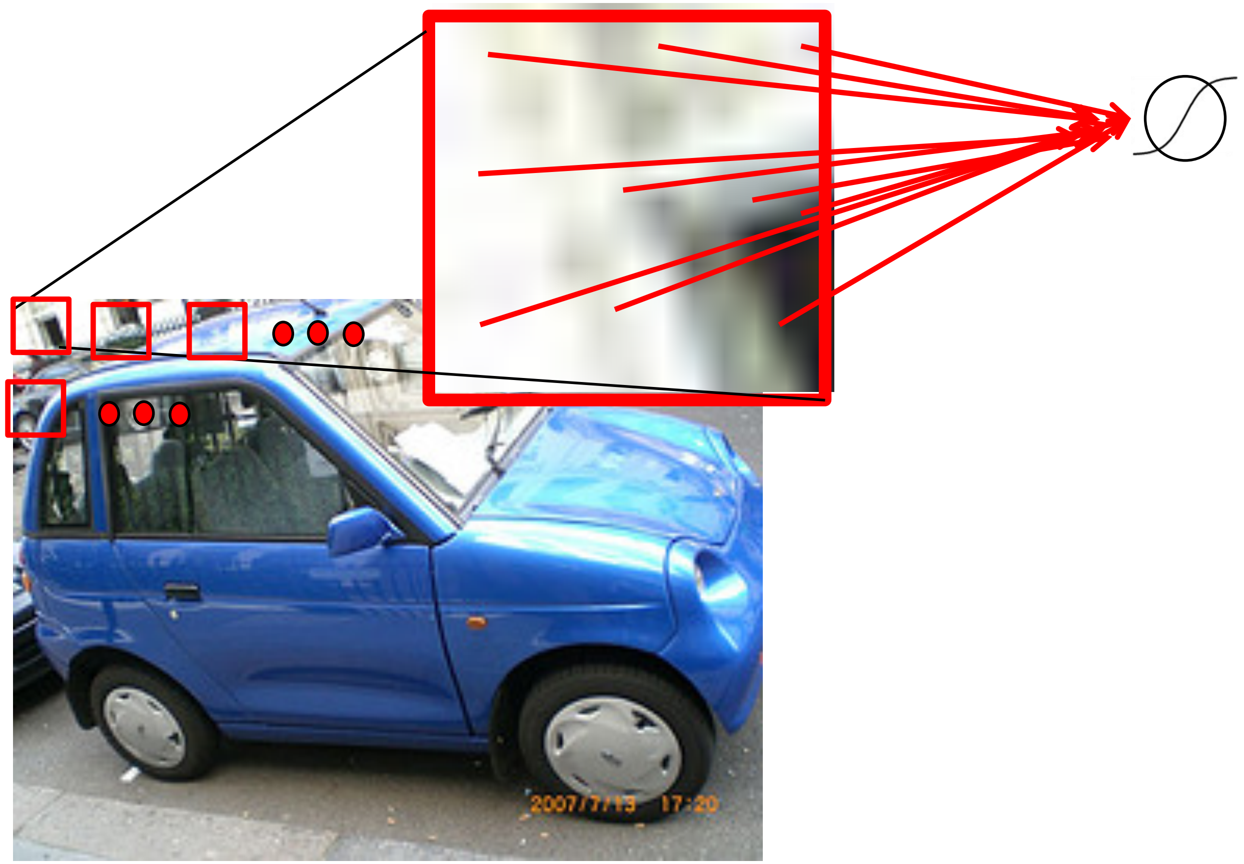
What W matrix would 'activate' for a upward-toward-left diagonal line?

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:

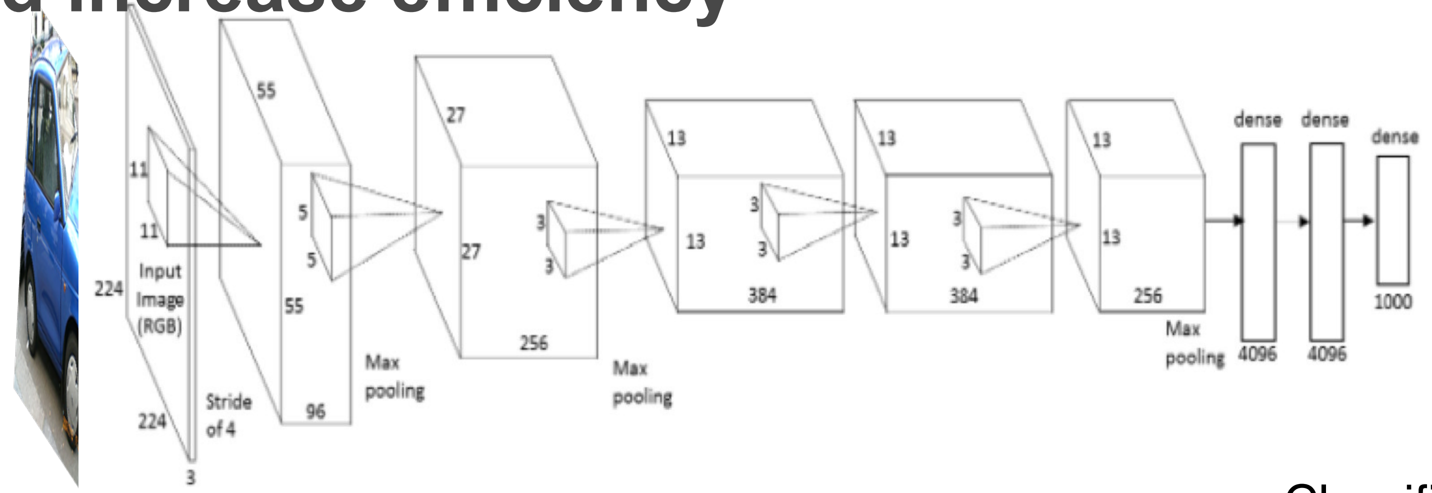
- filter size depends on input:
  - smaller filters for smaller details
  - 2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
  - smaller better but less efficient
- number of filters
  - depends on task
  - each filter is a new 2D layer

Convolution Network :

many layers and architecture options

Large Scale Versions

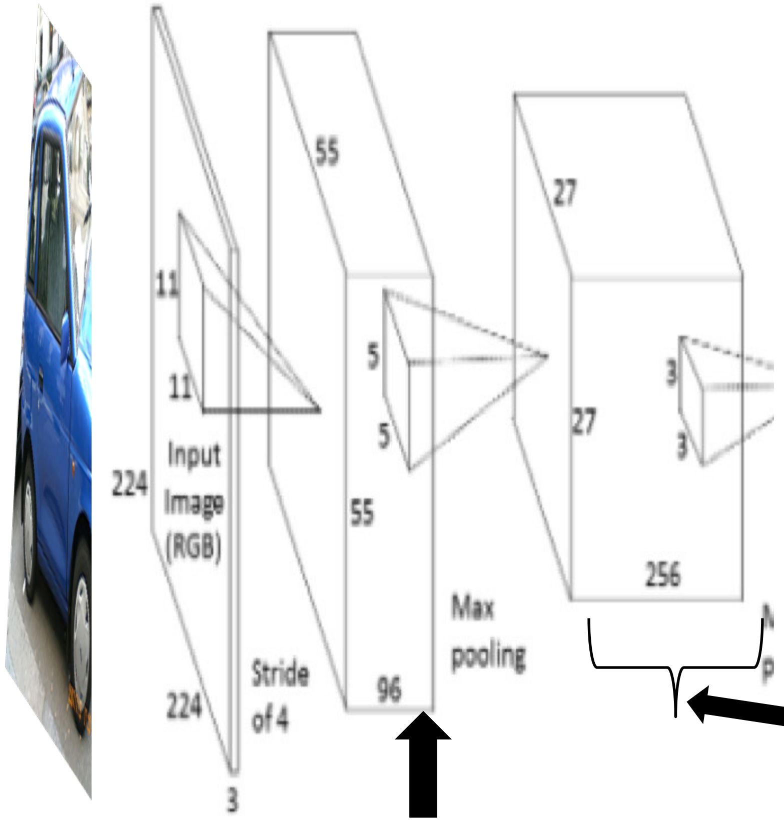
- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency



Convolution layers → Classification layers and output

Large Scale Versions

- **Zooming in:**

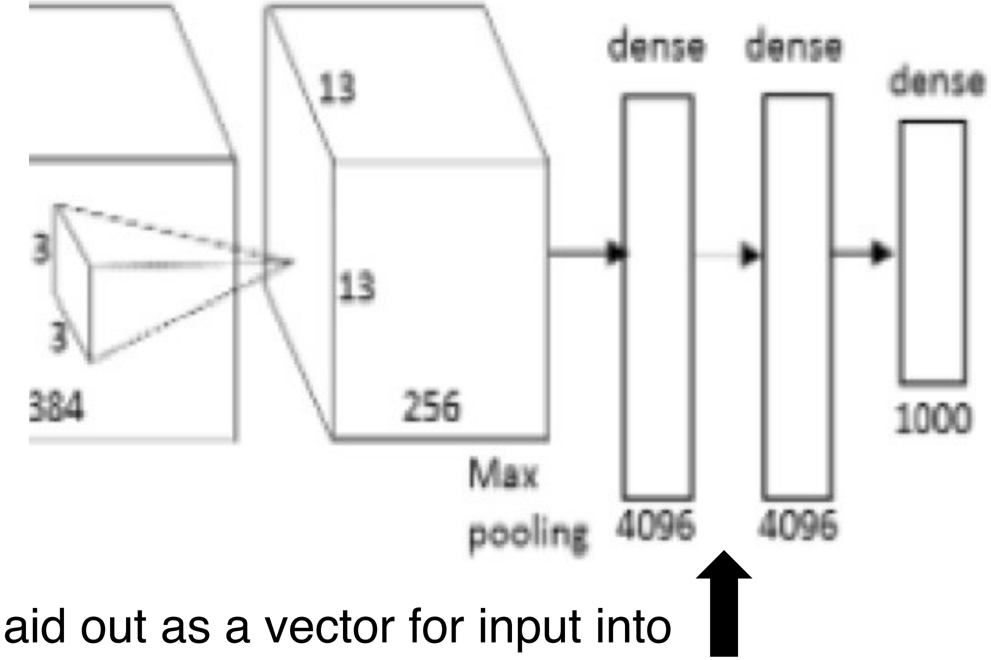


The thickness is the number of different convolutions, i.e. different transformations, sometimes called “channels”

Each convolution layer uses RELU (rectified linear activation units instead of logistic function) and is followed by Max Pooling layer (over 2D regions with sliding)

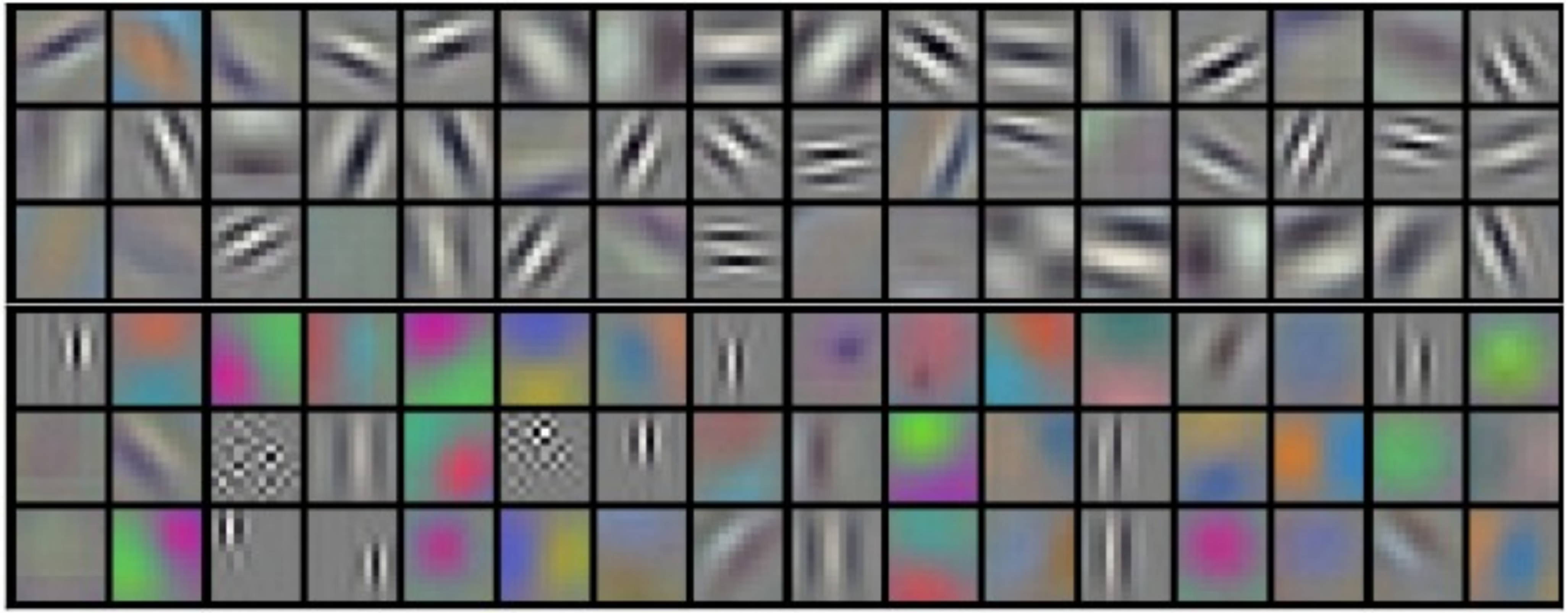
Large Scale Versions

- **Zooming in:**



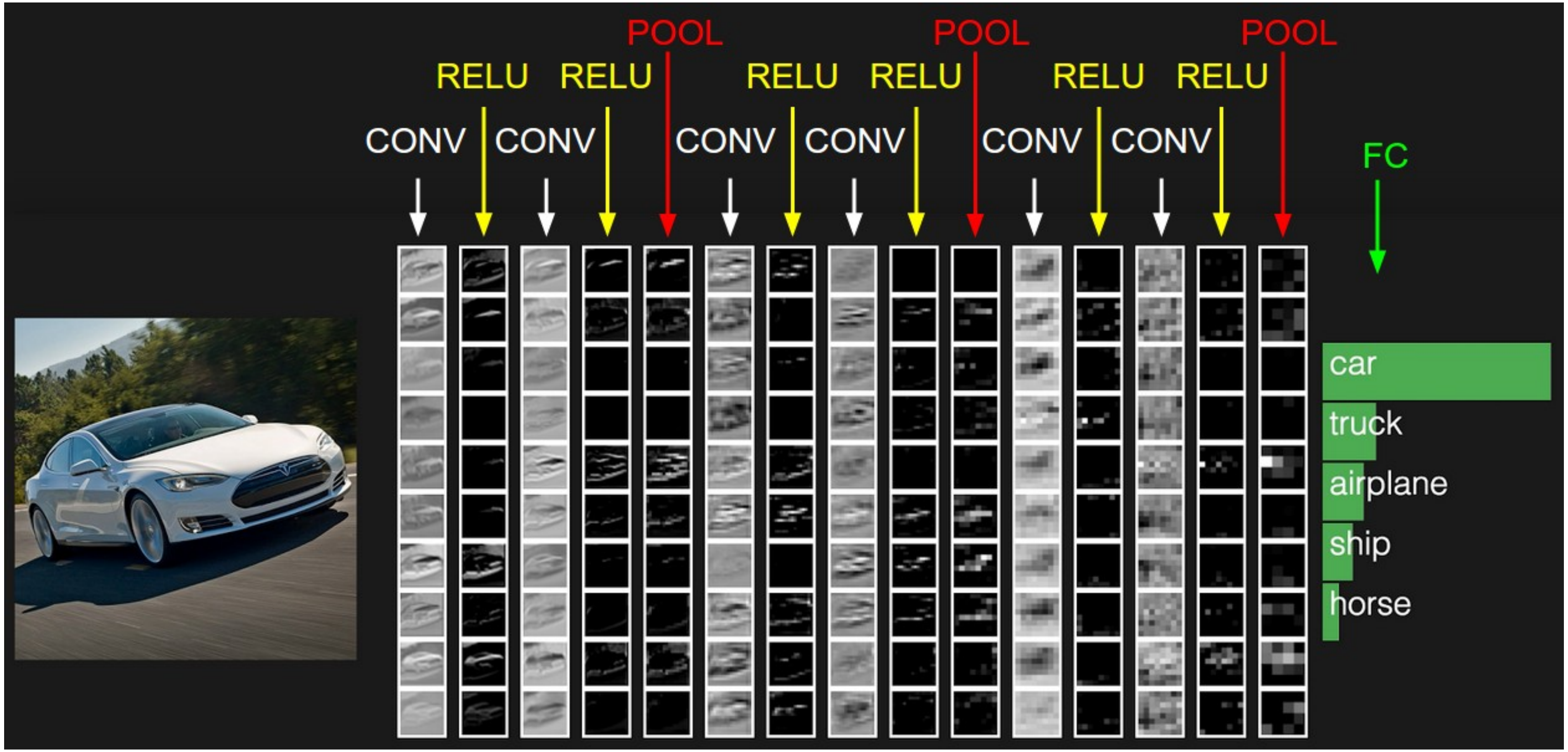
Last convolution layer is laid out as a vector for input into classification layers.
Classification uses dense, i.e. fully connected, hidden layers and output layer.

What Learned Convolutions Look Like



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." Advances in neural information processing systems. 2012.

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

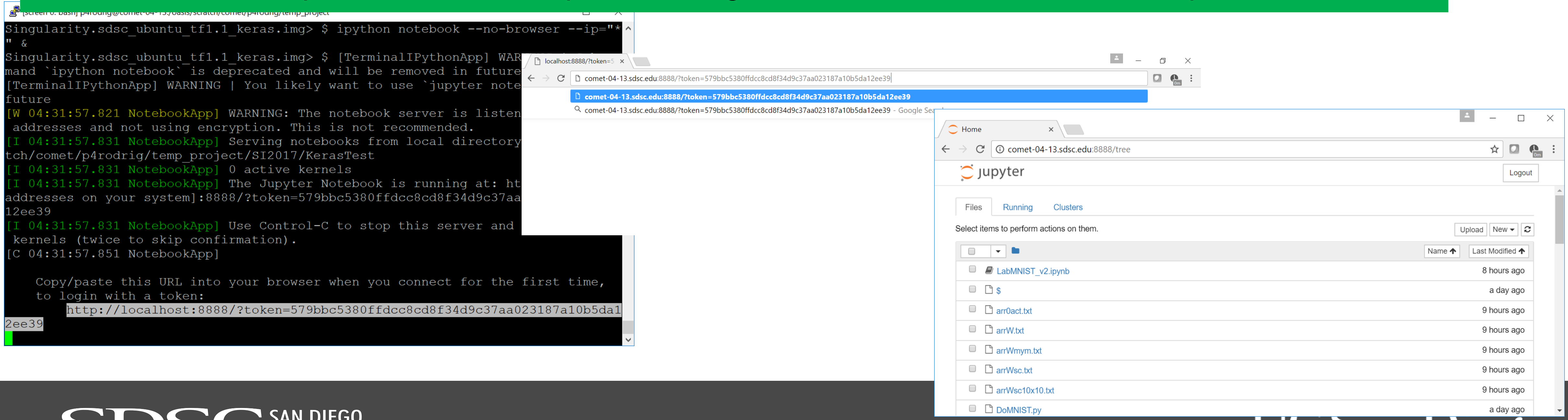
References

- **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**

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



LabMNIST_Final Last Checkpoint: 7 minutes ago (autosaved)

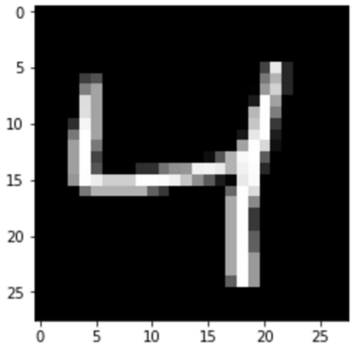
```
for i in range(0,3):
    im = Image.fromarray(X_train[i,:,:])
    im.save("Xtrain_num"+str(i)+"_cat_"+str(Y_train[i])+".jpeg")

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

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

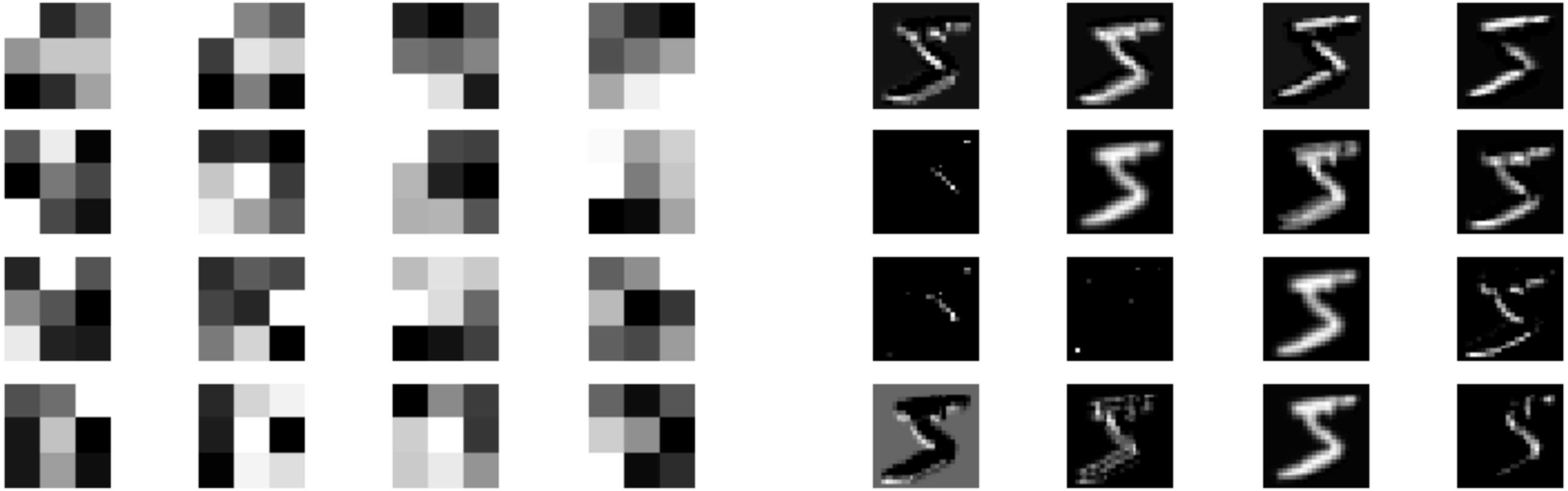
```
(5000, 28, 28)

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

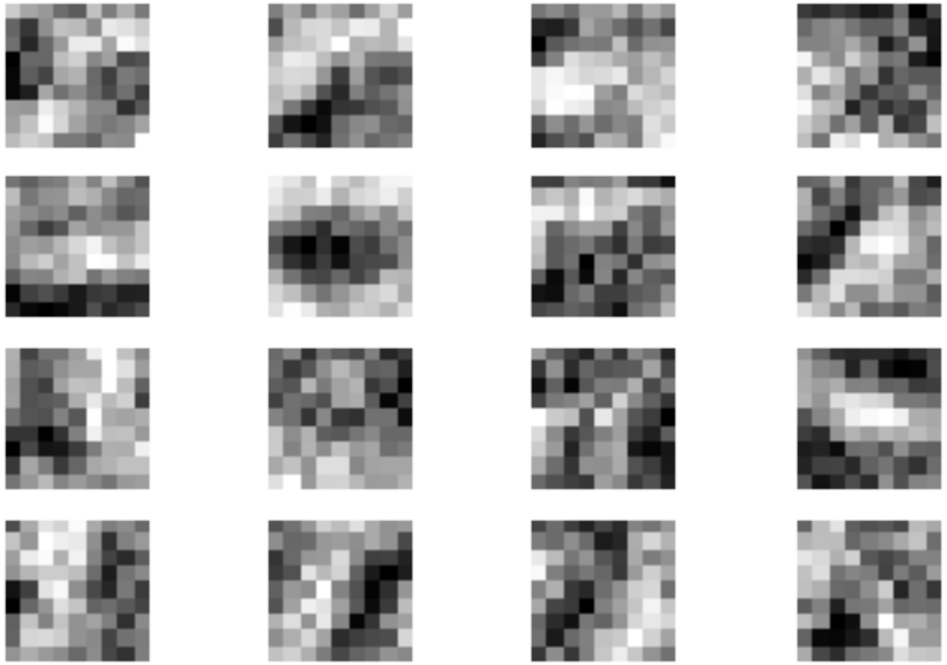


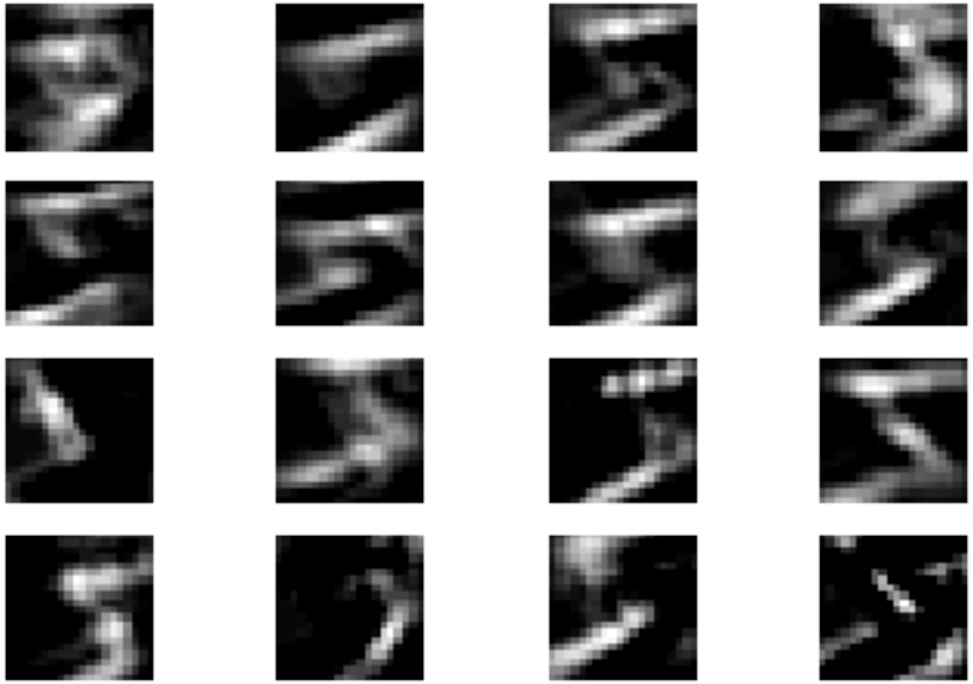

```
img load done
```

3x3 first convolution layer filter and activation



9x9 first convolution layer filter and activation





Finally…

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

Next- transfer learning