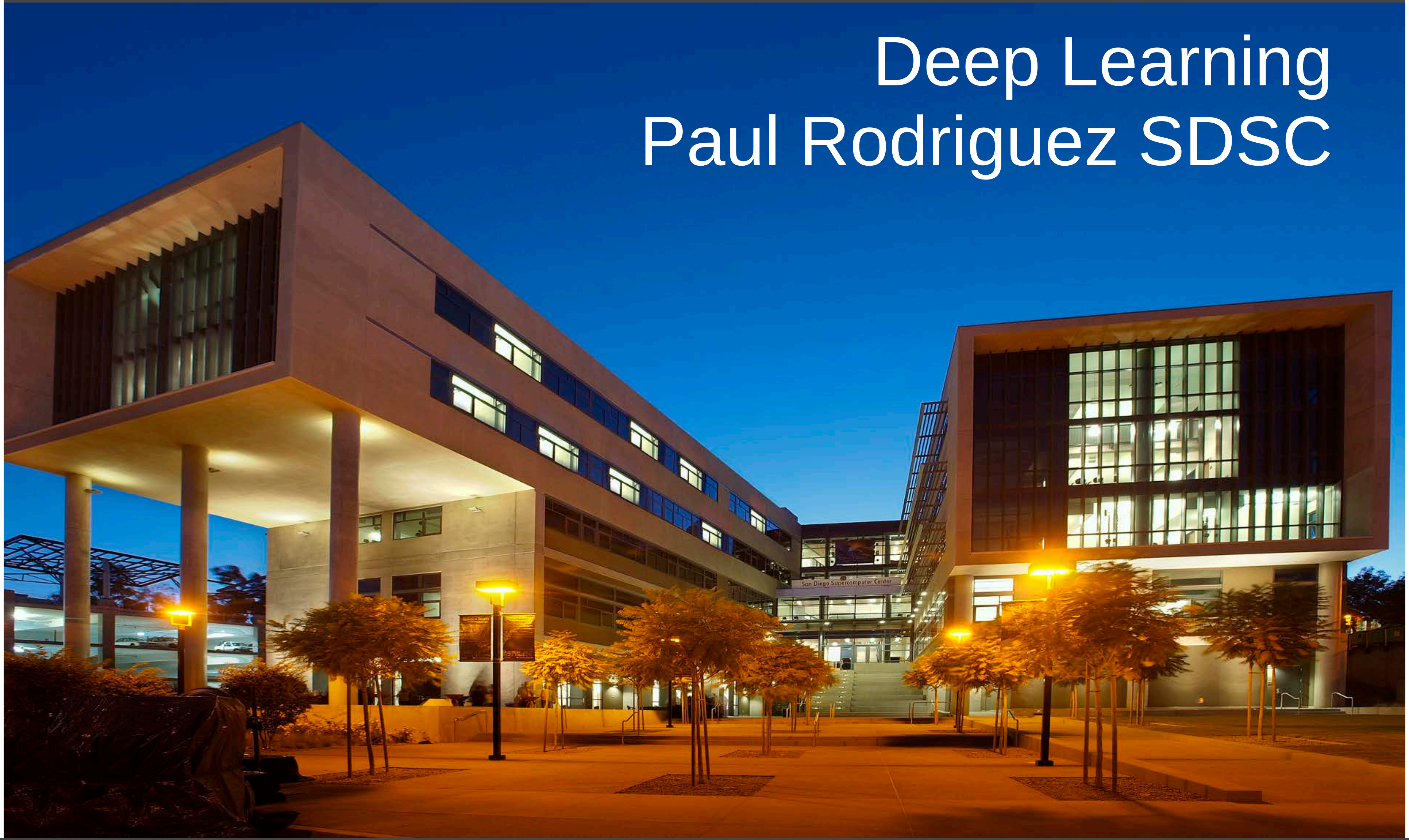
Deep Learning Agenda

1:30 – 2:07 Intro to Neural Nets/Convolution Nets
2:07 – 2:30 MNIST CNN tutorial
2:30 – 2:45 break
2:45 - 3:45 – CNN transfer learning & tutorial
3:45 - 4:15 – Object detection, fasterRCNN
4:15 – 4:45 – Object segmentation,
Sequence Learning
4:45 Wrap up

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. Discovering complex feature transformations

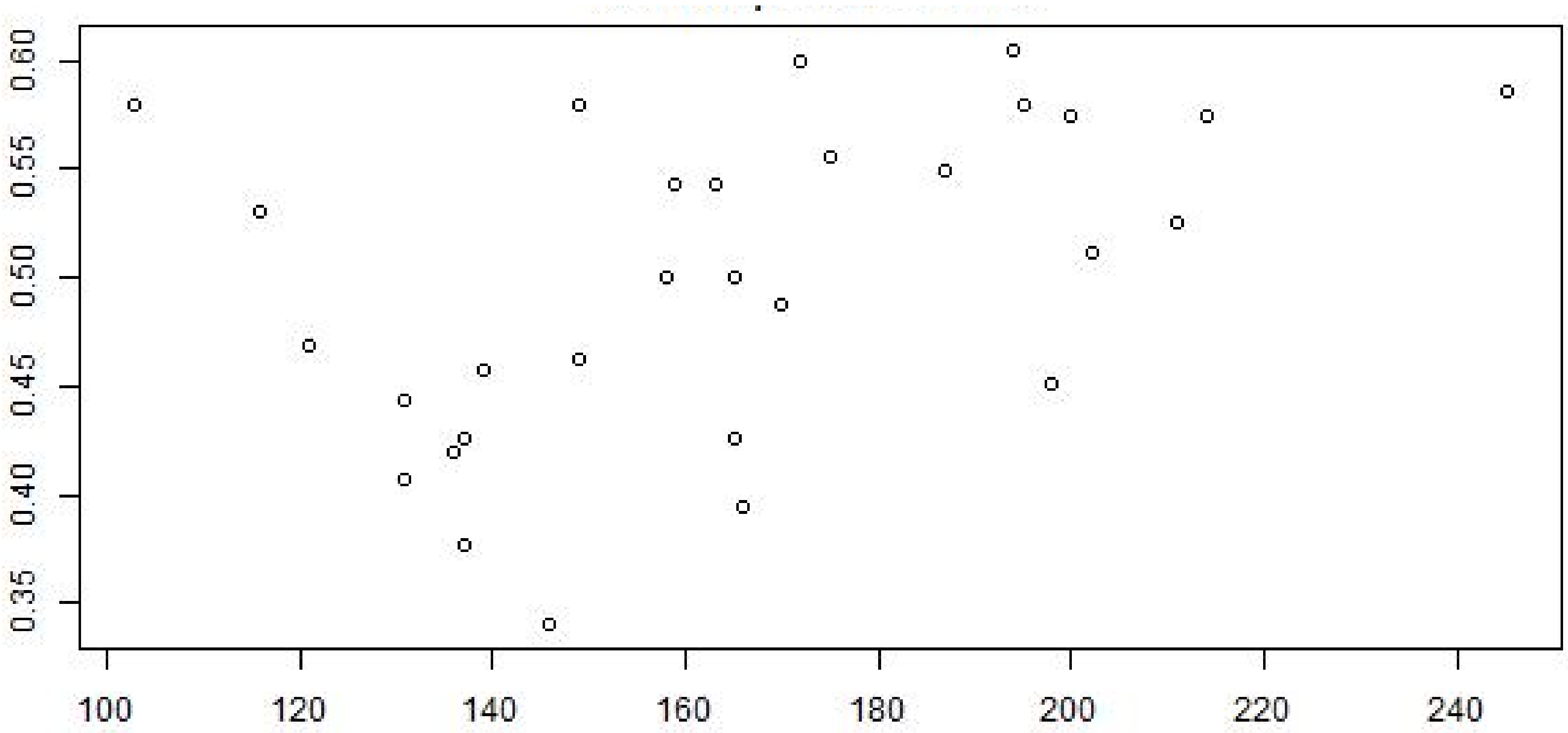
3. Using neural networks with many layers

Explanation Strategy: Start with linear regression and go deep

A data example:Home Runs and W-L percent

W-L %

(call it ‘*y*’)



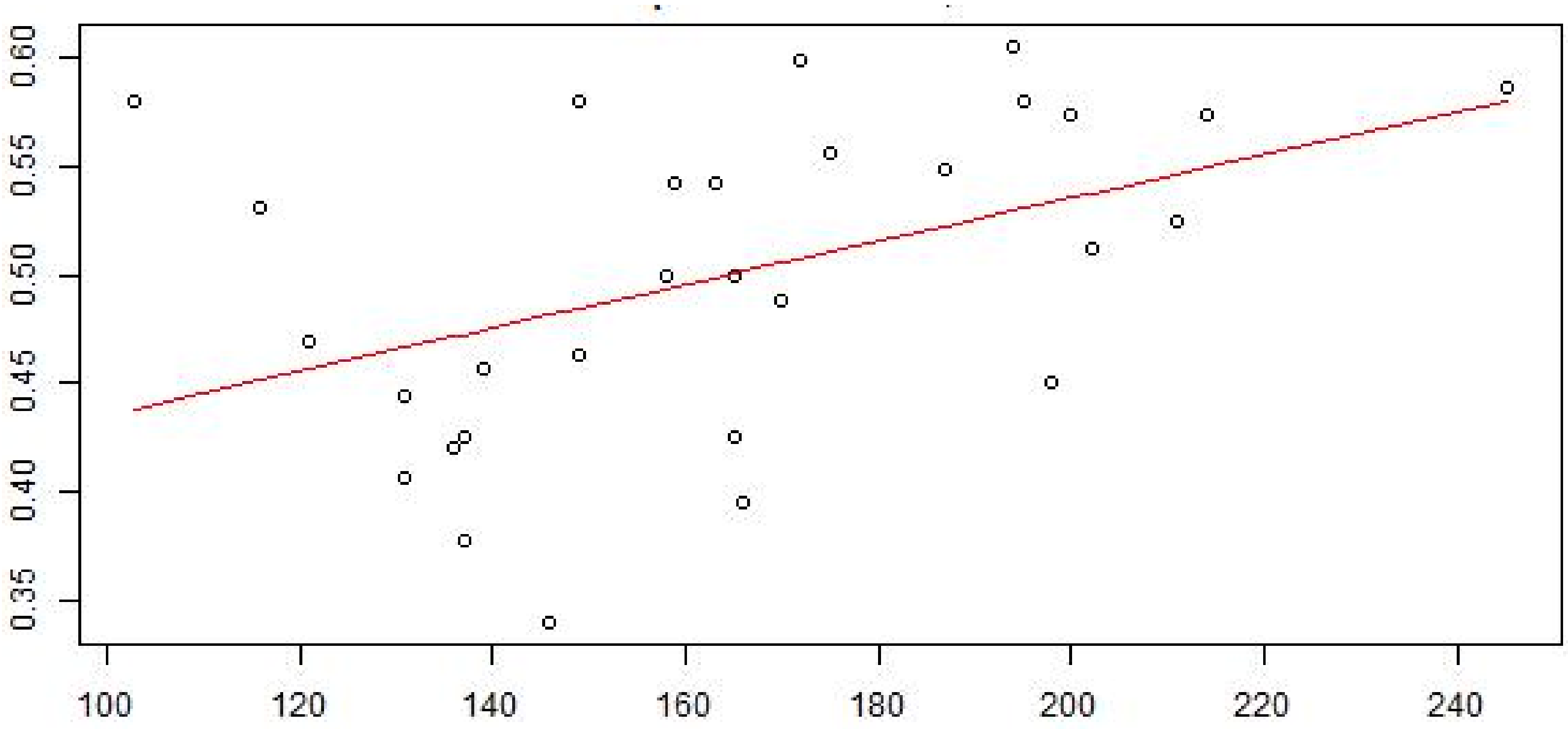
Number of Team HRs 2012 (call it ‘*x*’)

Recall Linear Regression is Fitting a Line

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

W-L %

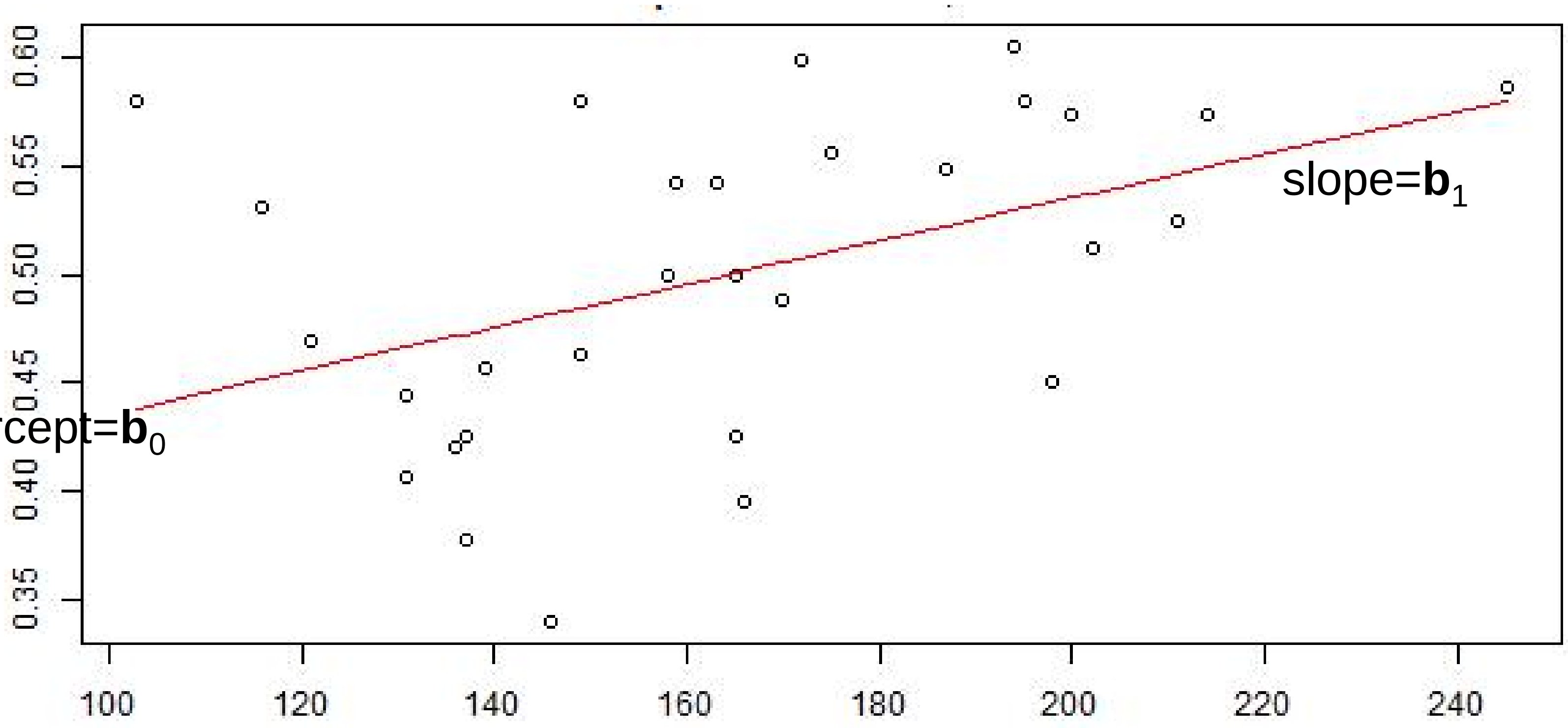
(call it '*y*')



Number of Team HRs 2012 (call it '*x*')

Recall Linear Regression is Fitting a Line

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



W-L %

(call it '*y*')

Intercept=$\mathbf{b}_0$

slope=$\mathbf{b}_1$

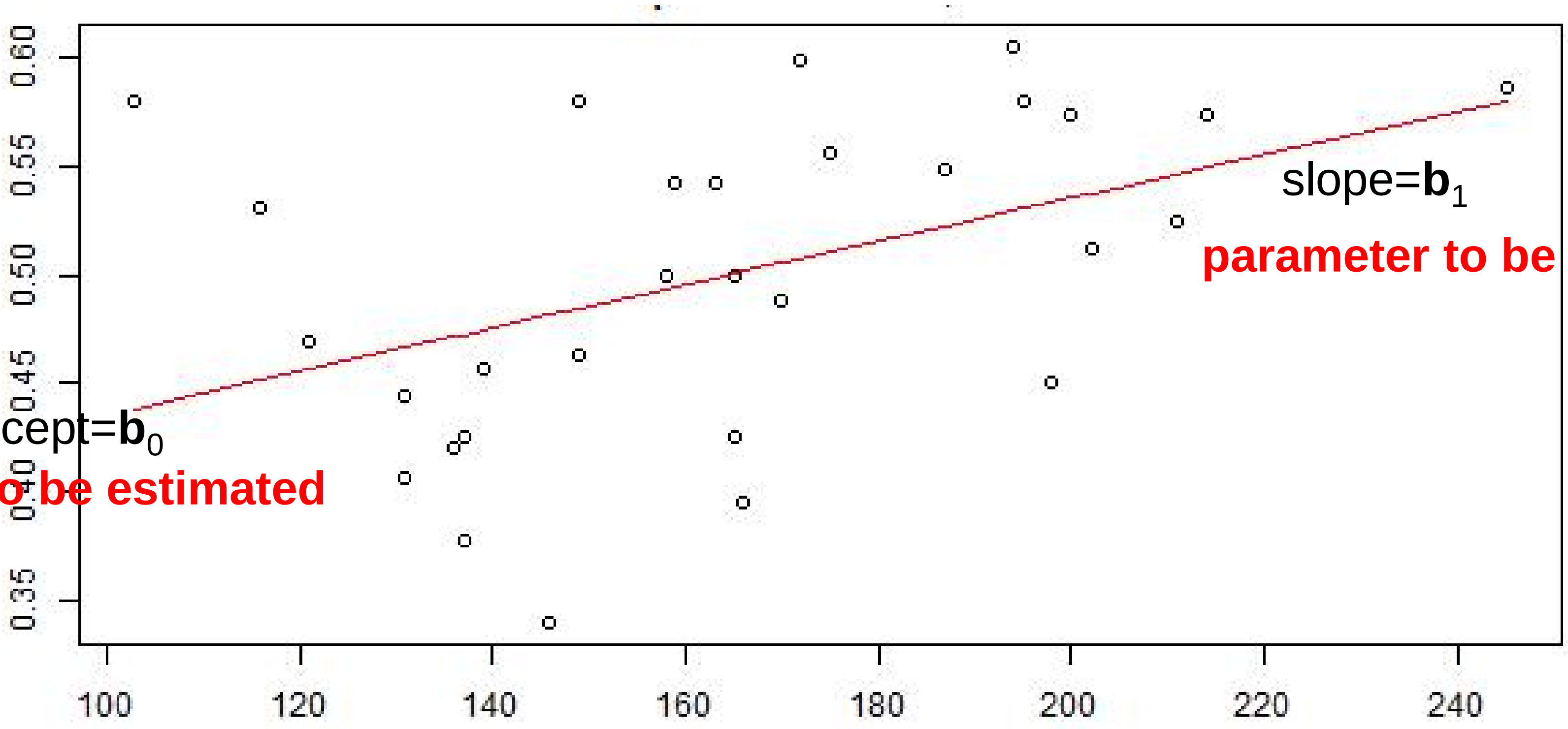
Number of Team HRs 2012 (call it '*x*')

Recall Linear Regression is Fitting a Line

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

W-L %

(call it '*y*')



slope=$\mathbf{b}_1$

parameter to be estimated

Intercept=$\mathbf{b}_0$

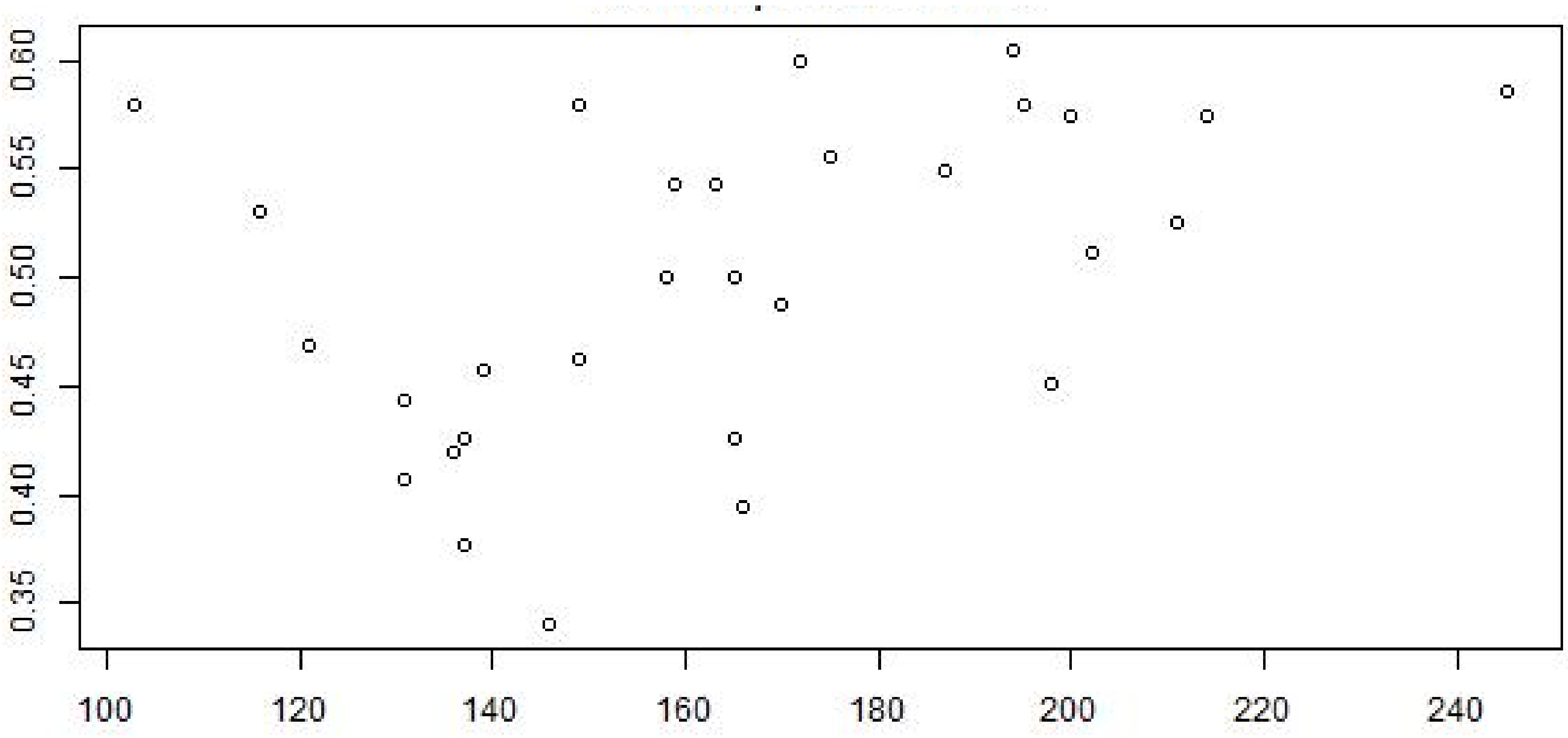
parameter to be estimated

Number of Team HRs 2012 (call it '*x*')

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

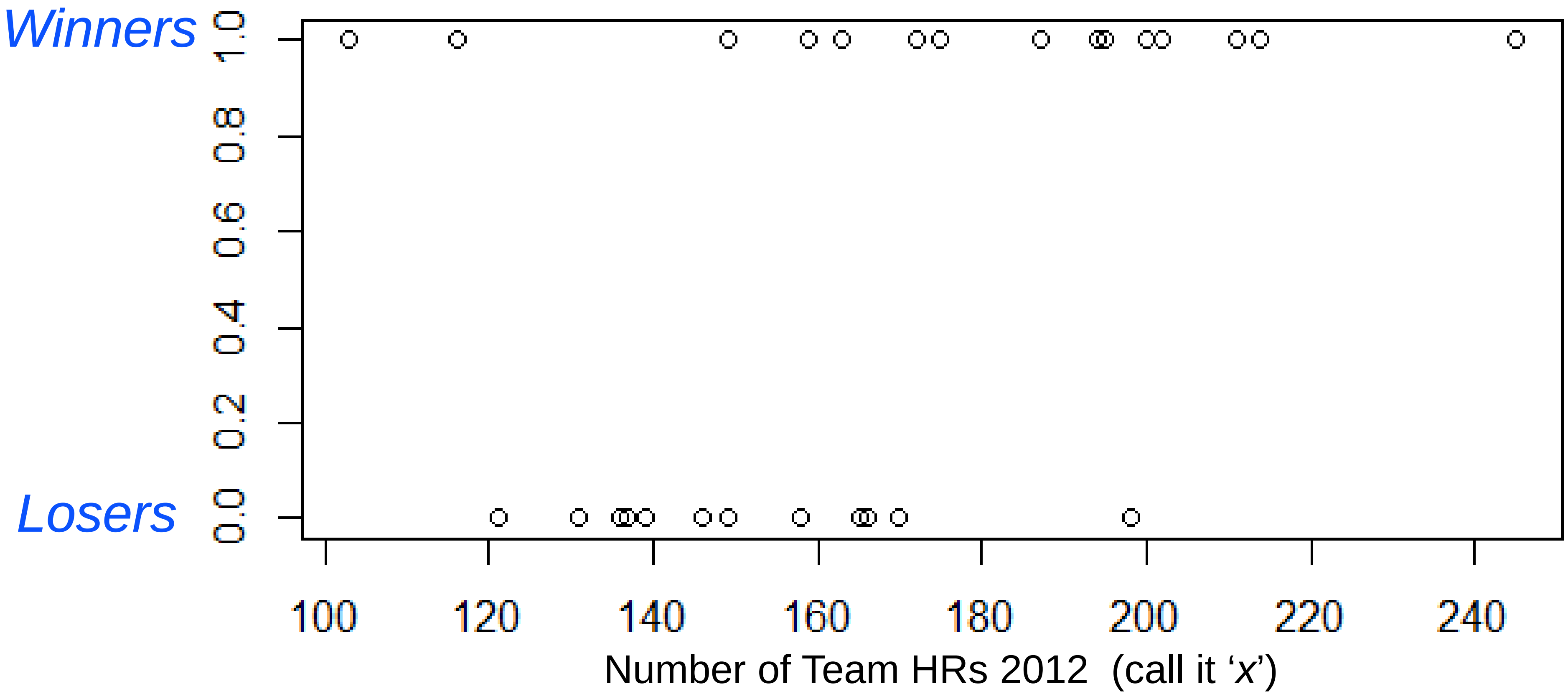
W-L %

(call it ‘*y*’)



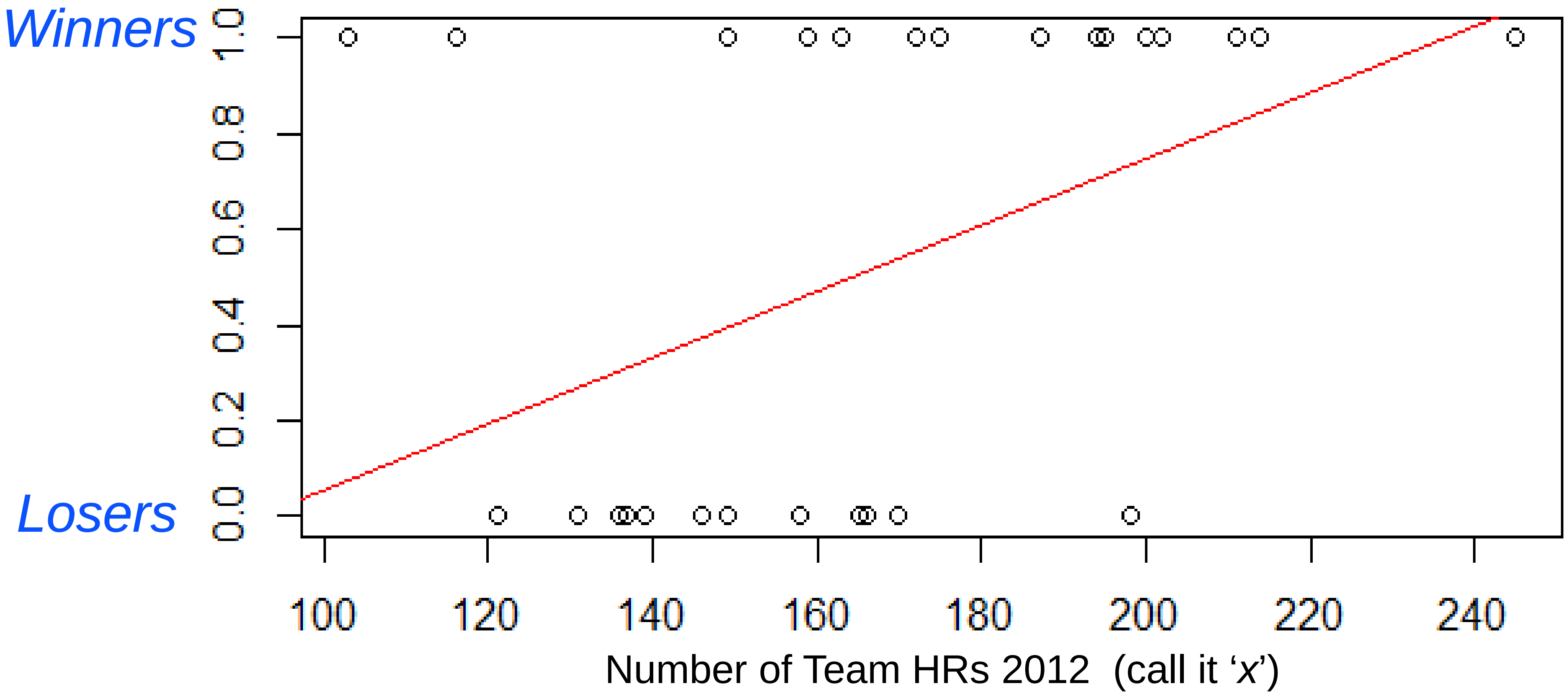
Number of Team HRs 2012 (call it ‘*x*’)

Classification uses labelled outcomes



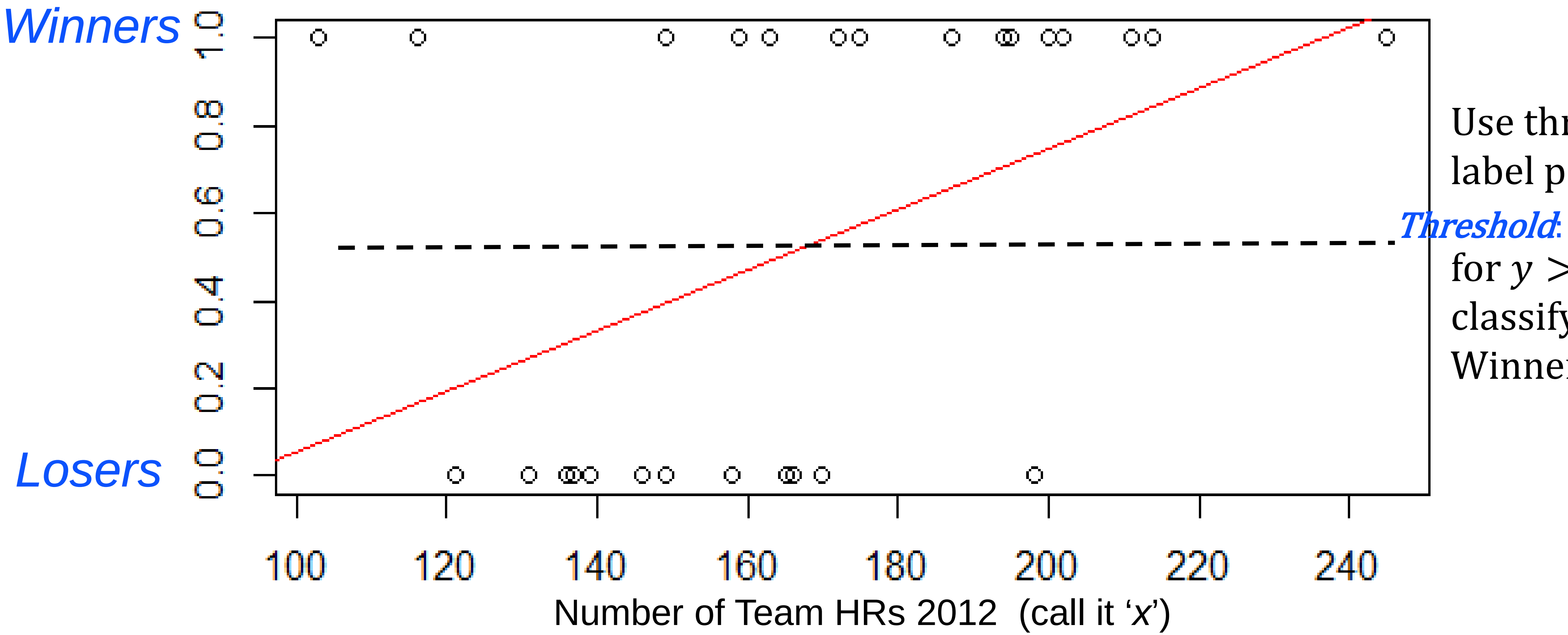
Can do this: fit a line with same model

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



Can do this: fit a line with same model

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

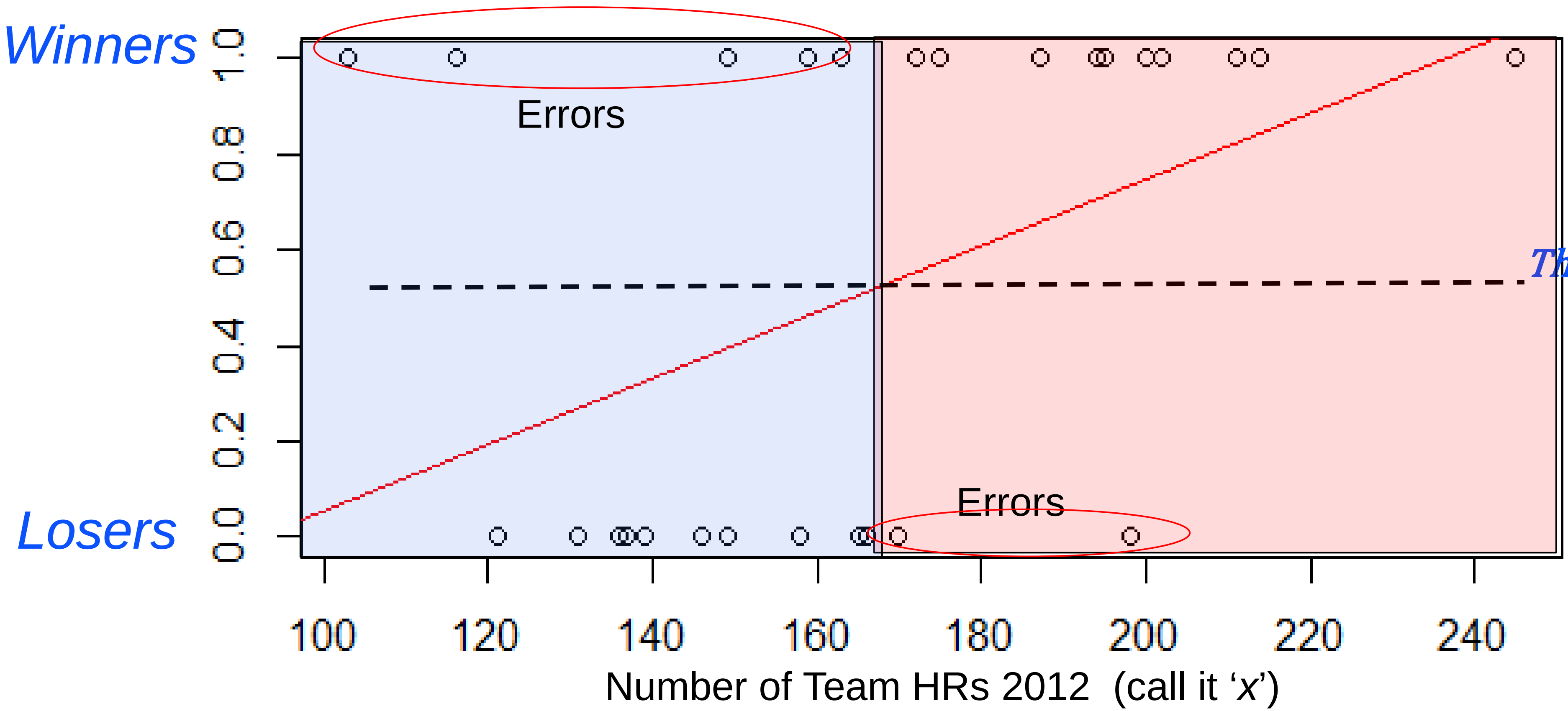


Use threshold to label prediction:

for $y > 0.5$, classify x as Winner

Can do this: fit a line with same model

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



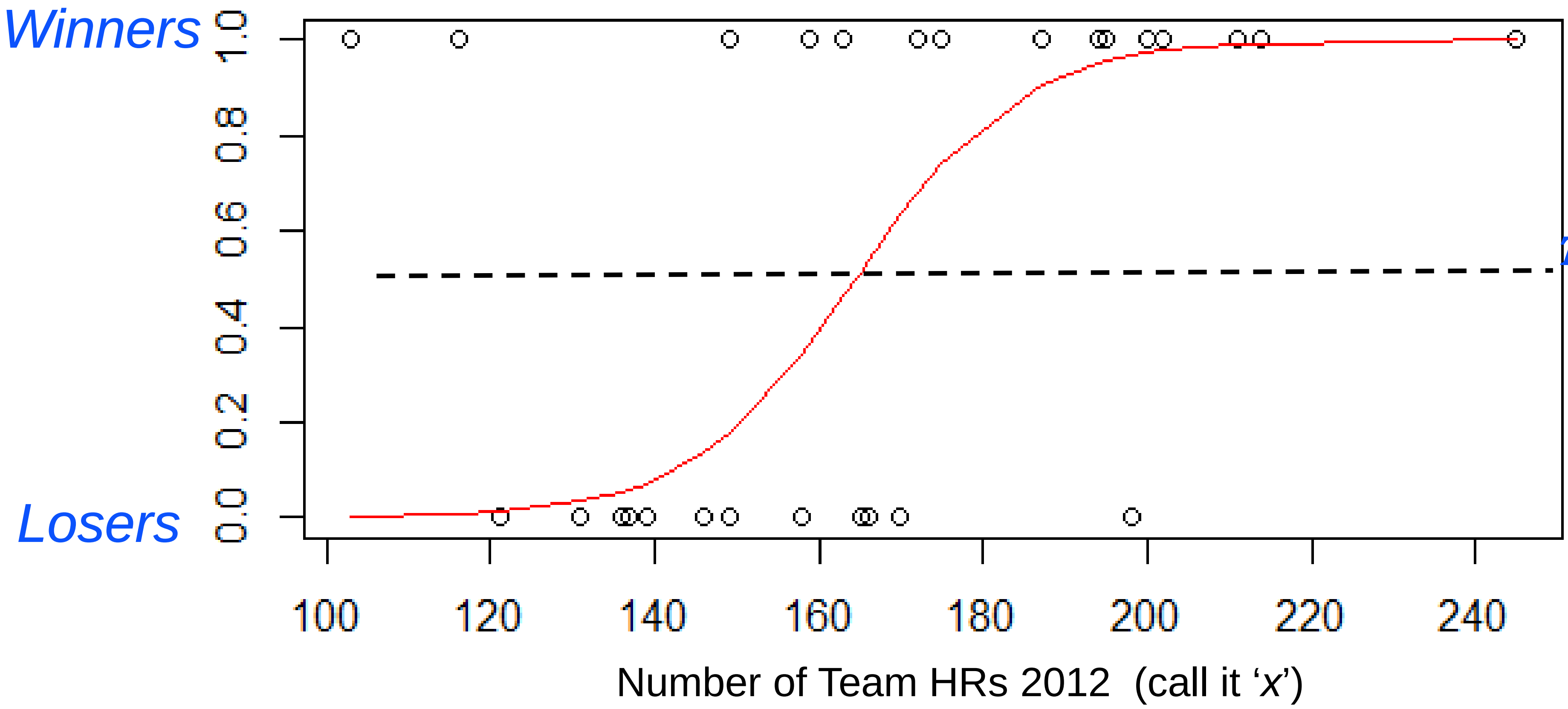
Use threshold to label prediction:

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

Divides X into Winner/Loser regions

Can do better: fit a nonlinear function

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

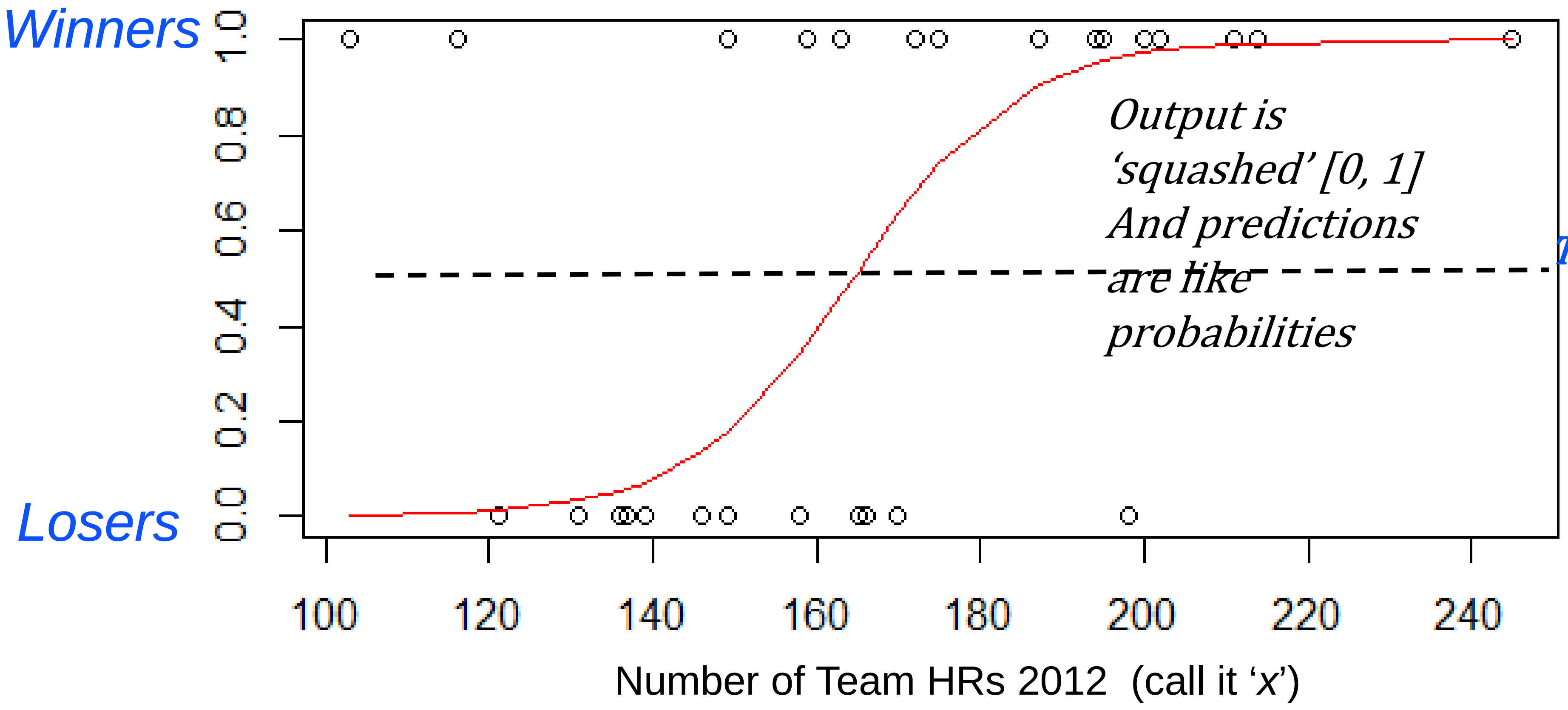


Use threshold to label prediction:

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

Can do better: fit a nonlinear function

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



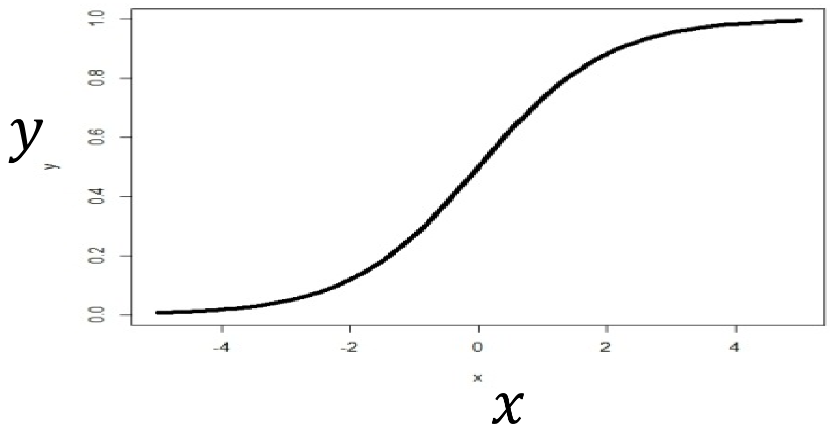
Use threshold to label prediction:

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

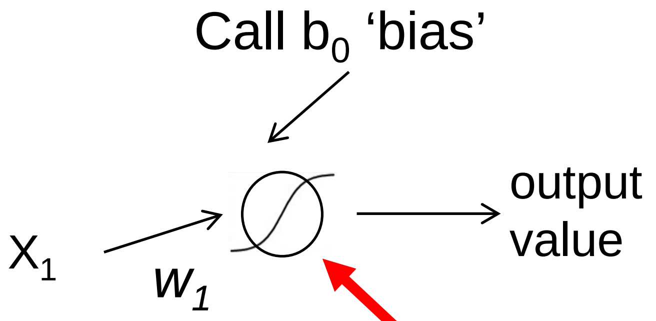
Logistic to Neural Network model

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

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



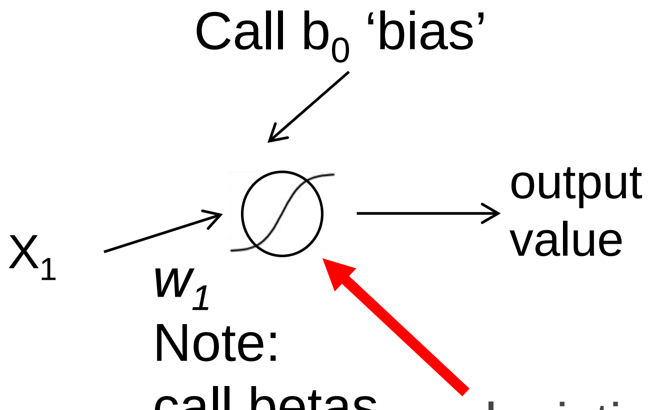
Logistic Regression as 1 node network



Note:
call betas
weights

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

Logistic Regression as 1 node network

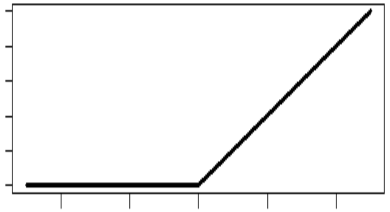


Note:
call betas
weights

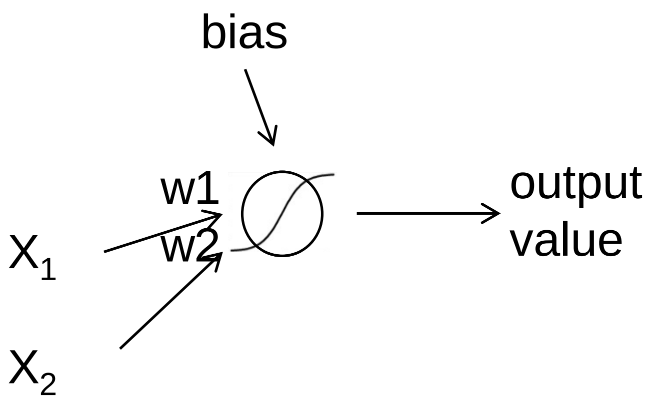
logistic function to transform input to output – call it the 'activation' function

Note: other activations are possible,

RELU (rectified linear unit)



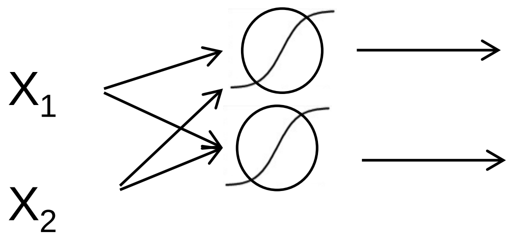
Next step: More general networks



Add input variables

More general networks

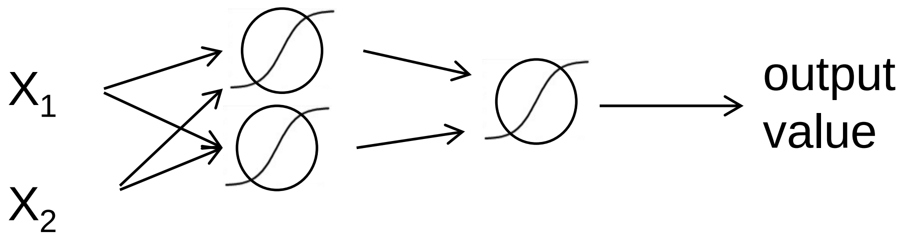
(assume bias present)



Add input variables *Add logistic transformations …*

More general networks

(assume bias)



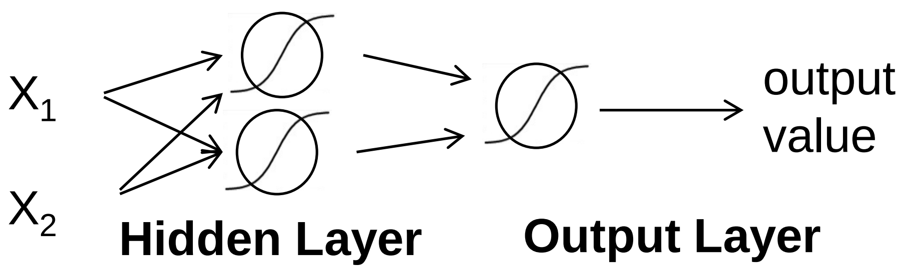
Add input variables

Add logistic transformations …

Combine transformations!

More general networks

(assume bias)



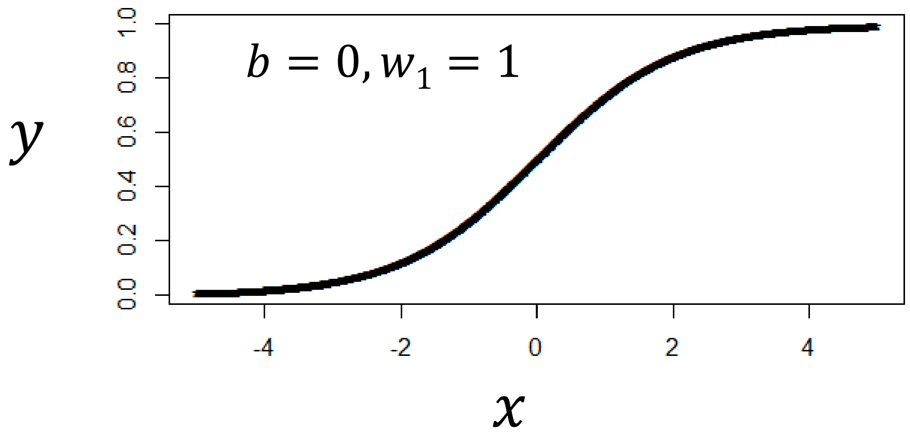
Add input variables

Add logistic transformations …

Combine transformations!

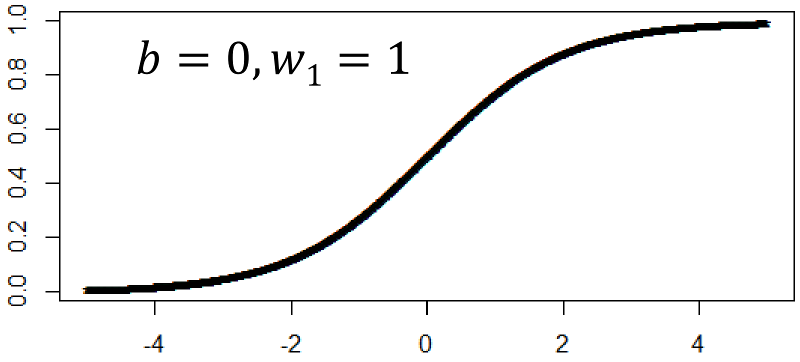
Logistic function w/various weights

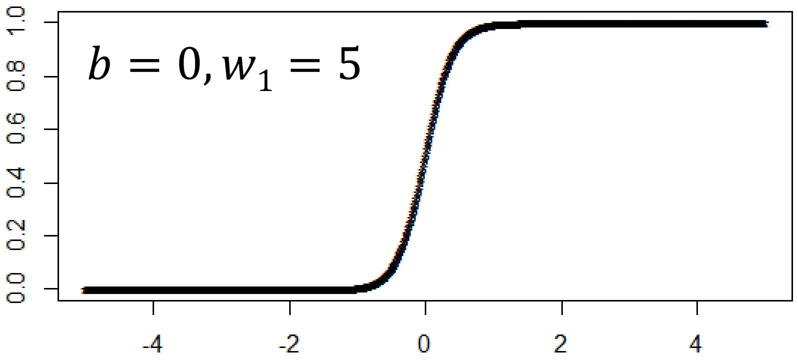
$$for\ y = 1/(1+exp(-(b+w_1 * x))$$



Logistic function w/various weights

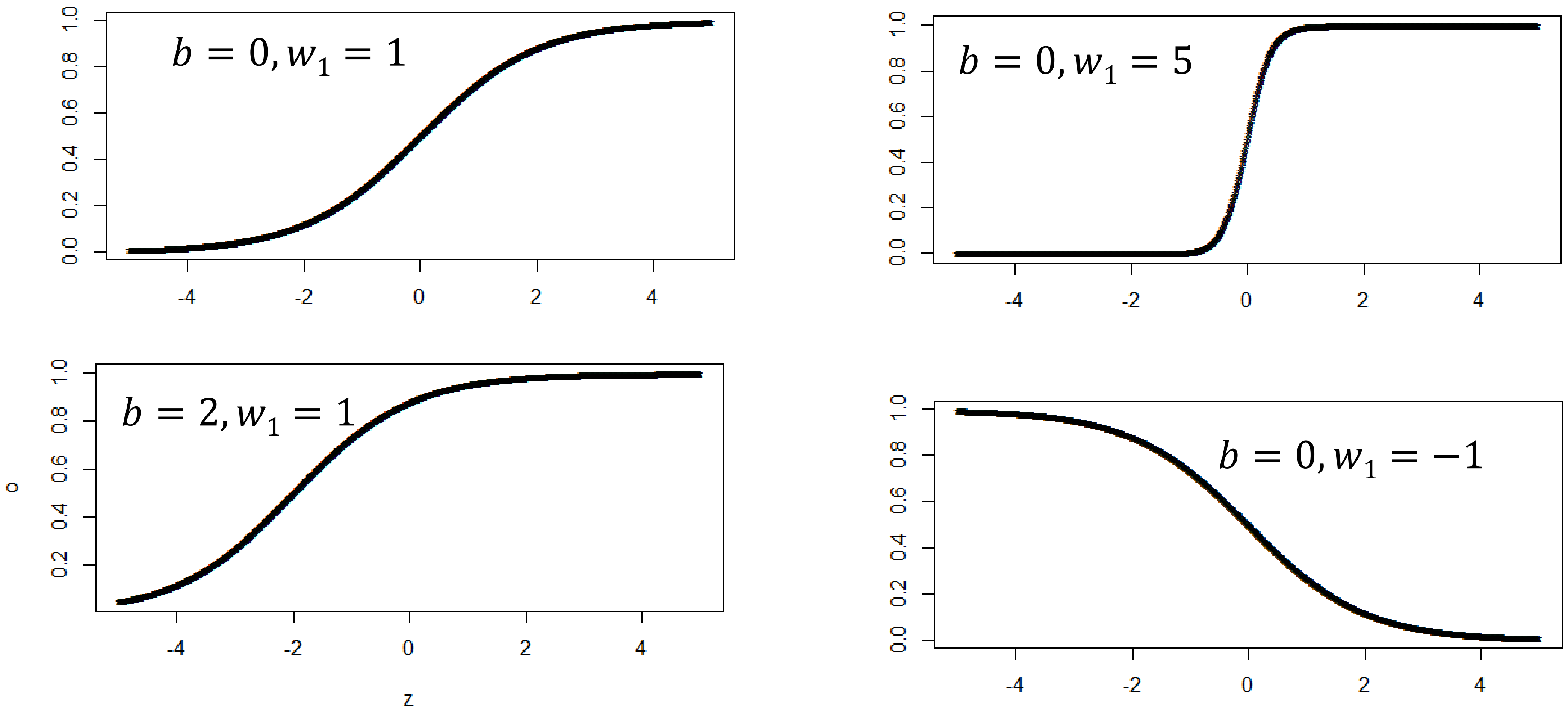
$$for\ y = 1/(1+exp(-(b+w_1*x))$$



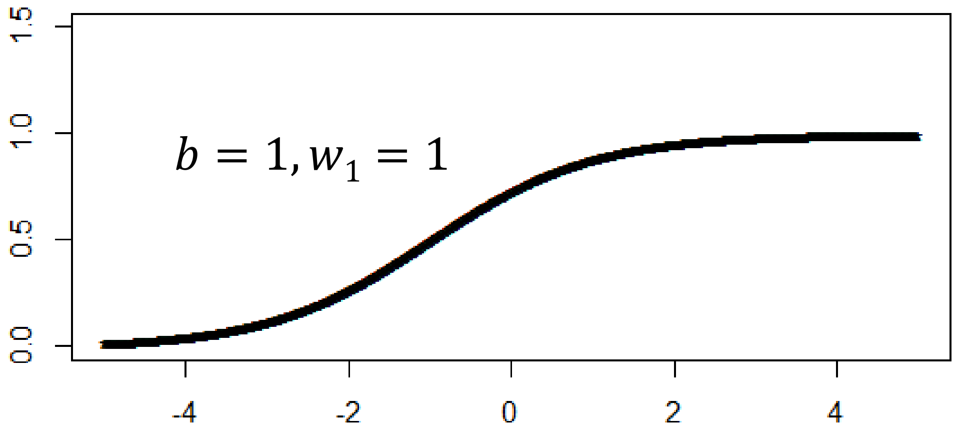


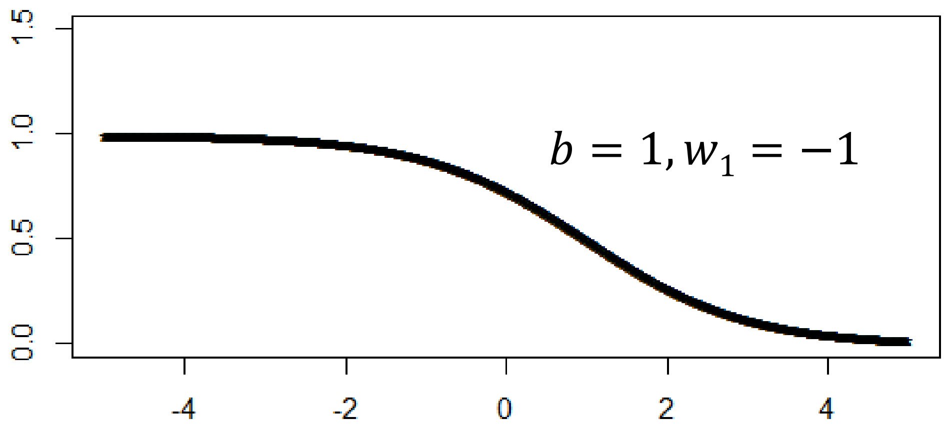
Logistic function w/various weights

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

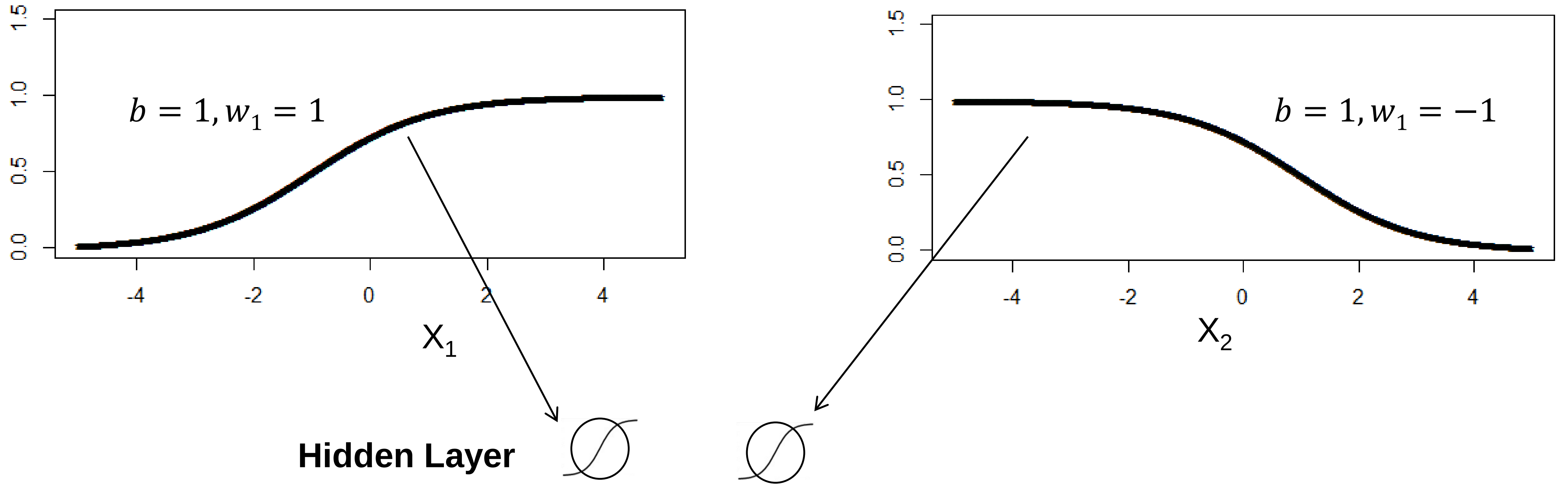


So combinations are highly flexible and nonlinear

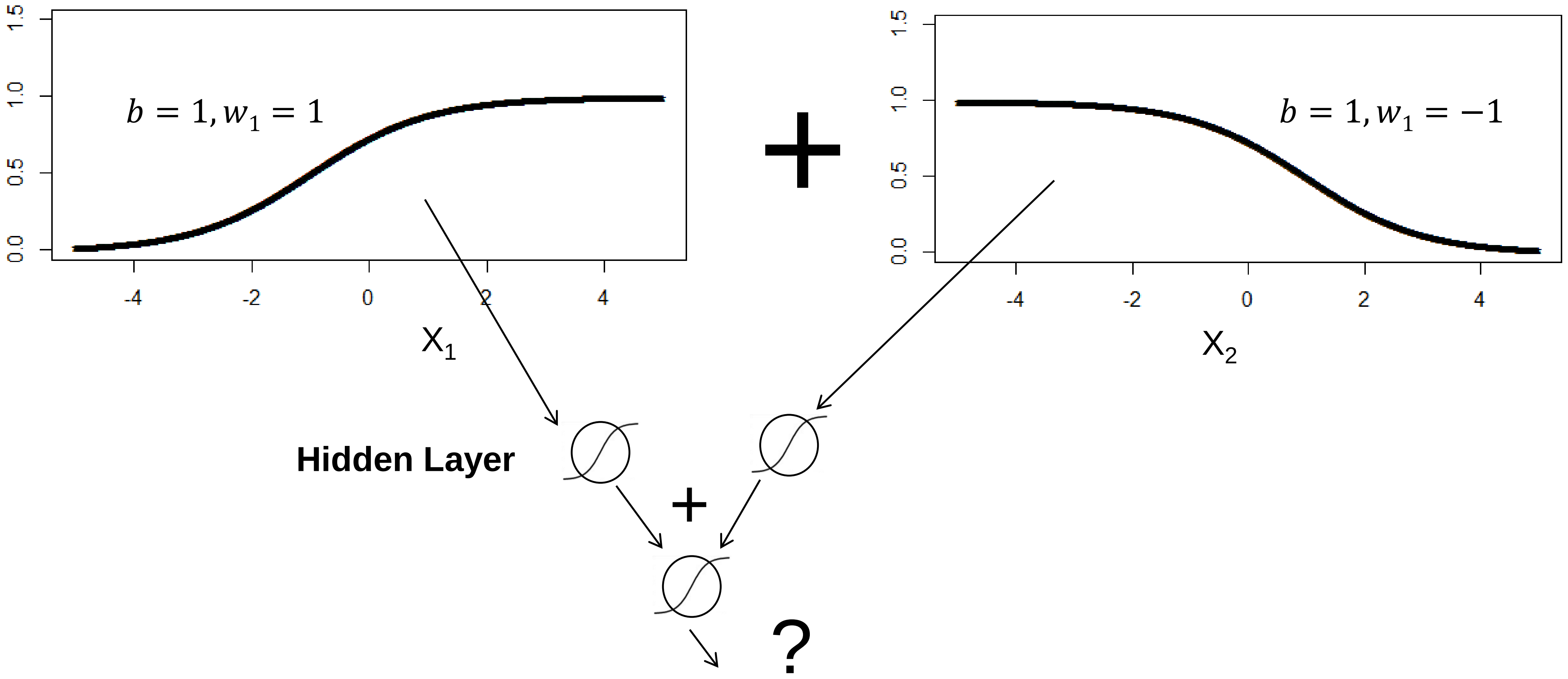




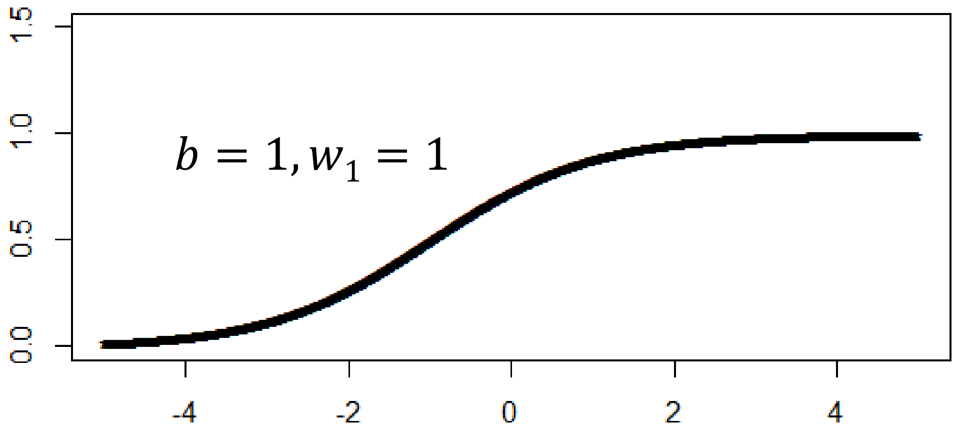
So combinations are highly flexible and nonlinear



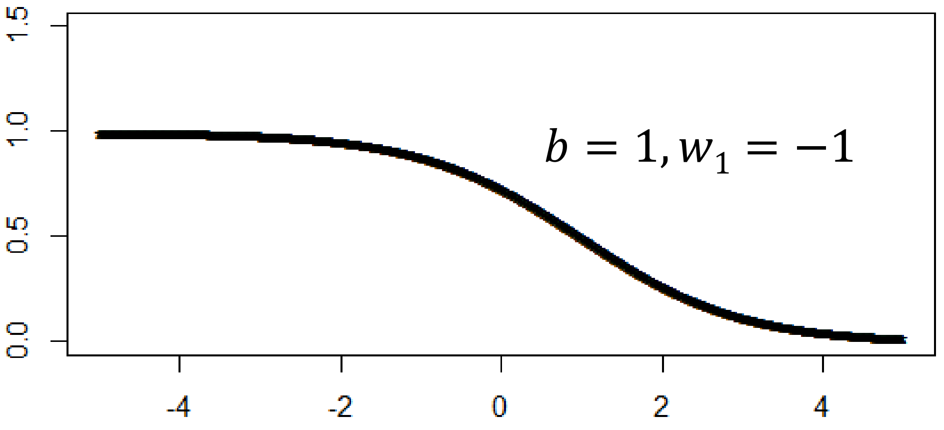
So combinations are highly flexible and nonlinear



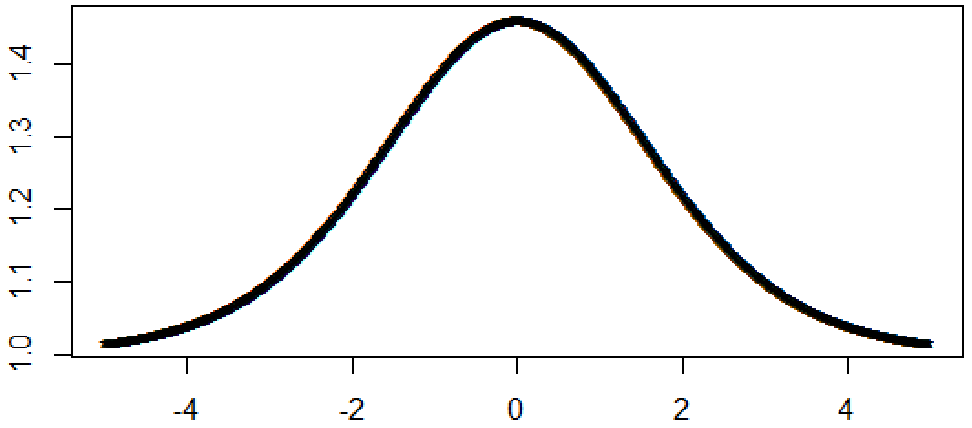
So combinations are highly flexible and nonlinear



+

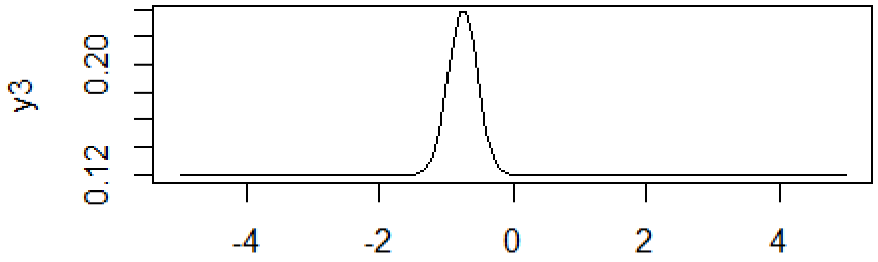


=

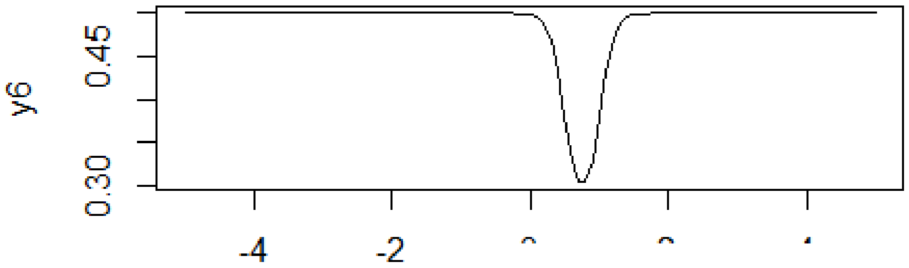


More function combinations

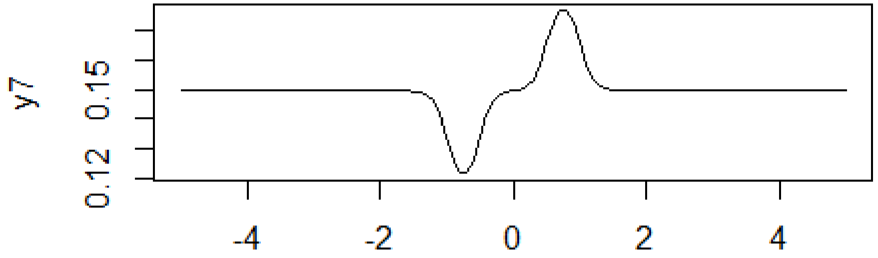
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```

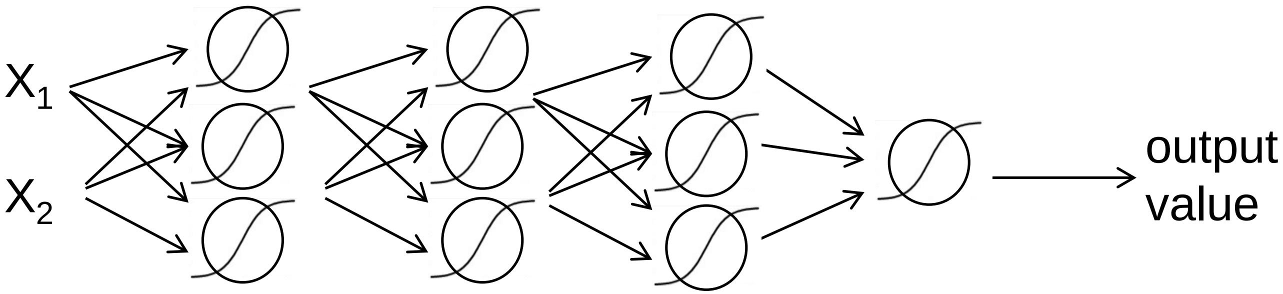



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```

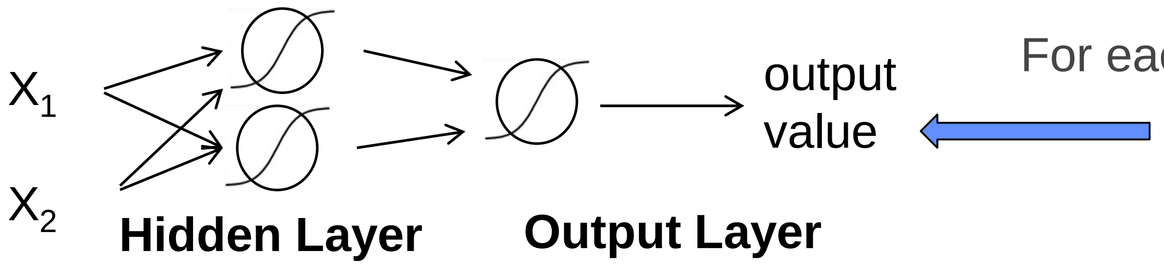


Why stop at 1 hidden layer?

More hidden layers => More varied features and transformation



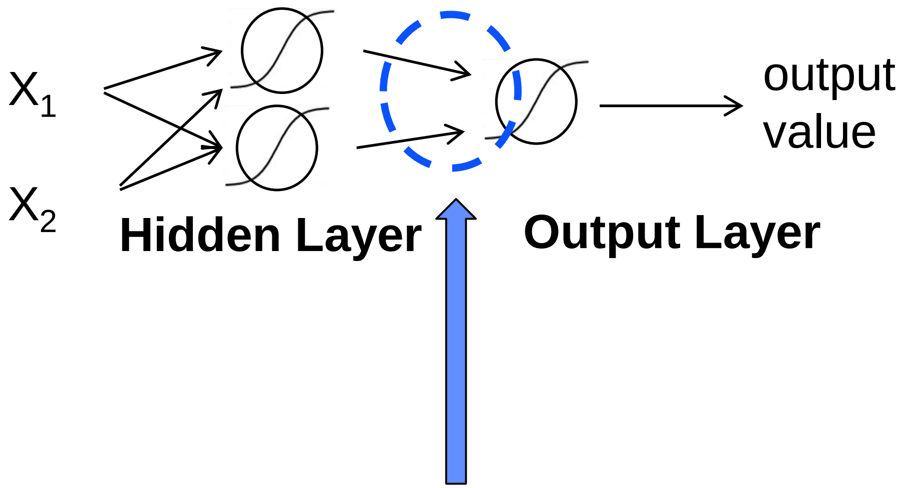
But parameter fitting is harder too



For each data instance:

Error=Output – Target

But parameter fitting is harder too

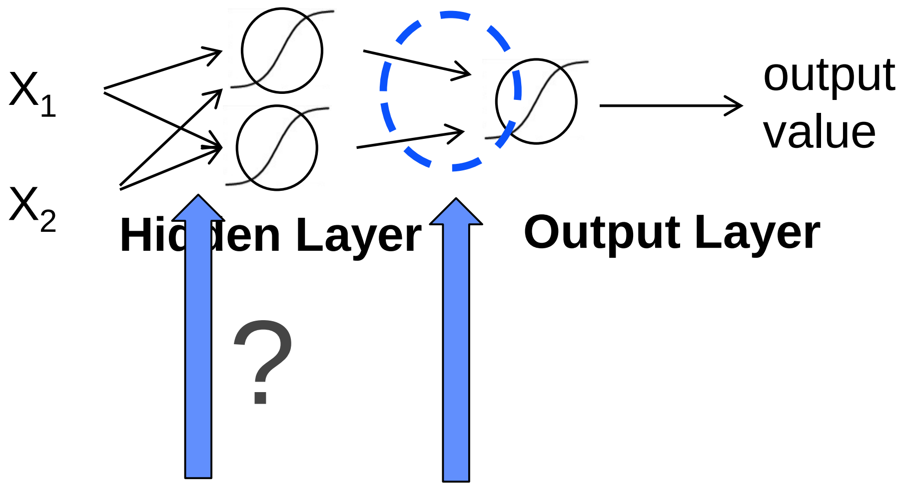


For each data instance:

Error=Output – Target

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

But parameter fitting is harder too



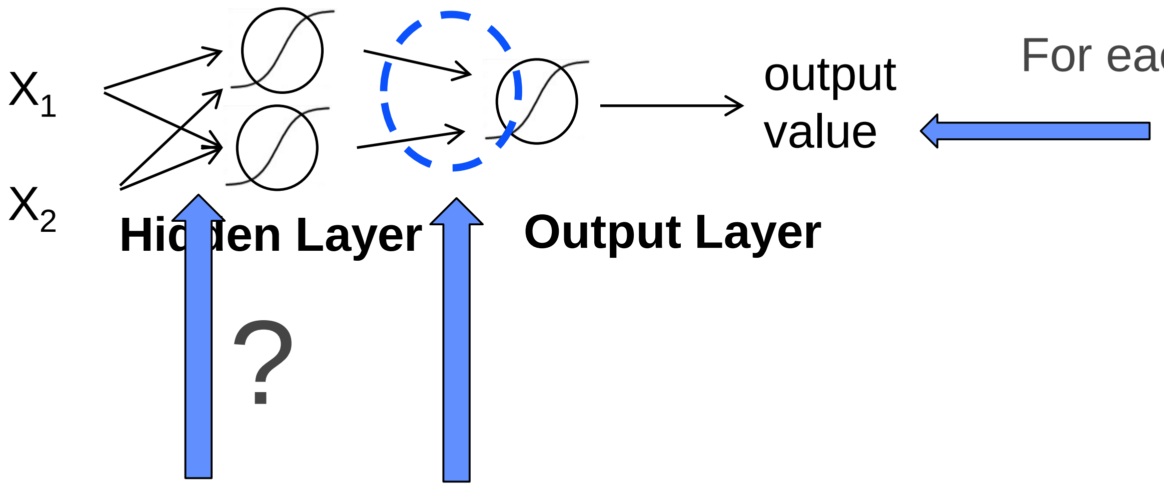
For each data instance:

Error=Output – Target

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

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 are only known for output layer, what is error for hidden layer?

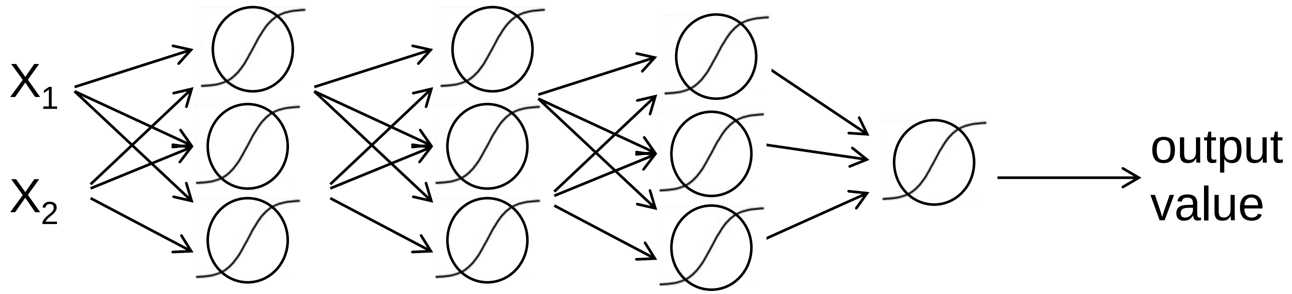
The objective is to minimize *Error* related to output weights (same as for logistic regression)

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

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

Train with Care

More hidden layers => More varied features and transformations

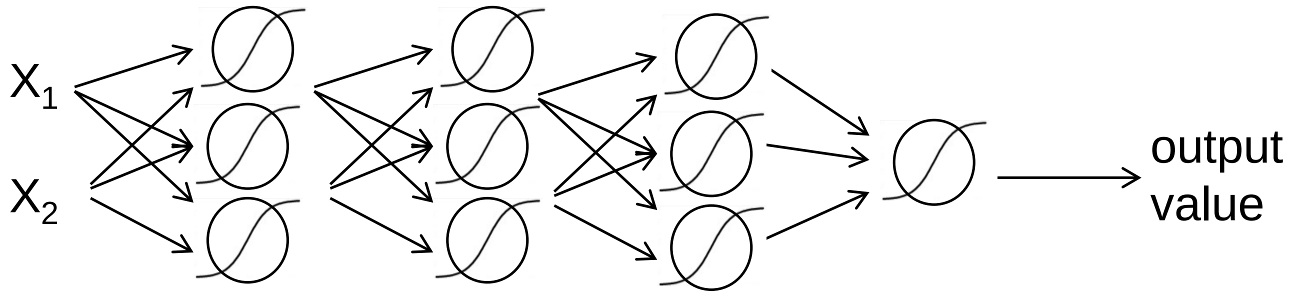


But:

More layers => more parameters

Train with Care

More hidden layers => More varied features and transformations



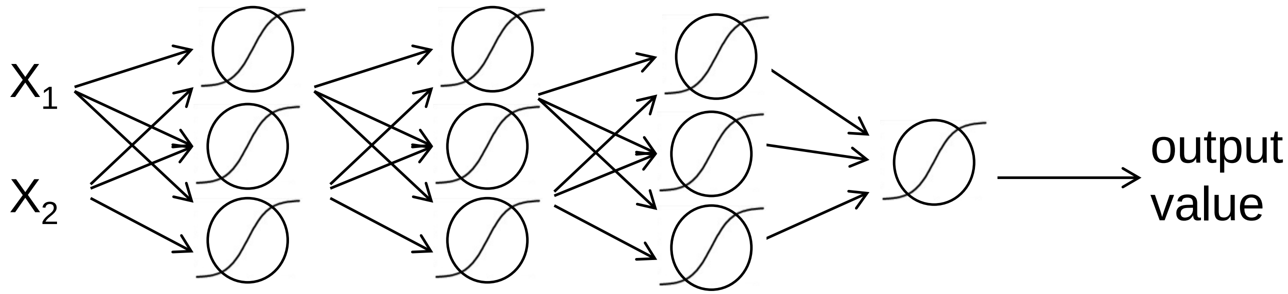
But:

More layers => more parameters => Smaller error for each

***especially* at lower layers**

Train with Care

More hidden layers => More varied features and transformations



But:

More layers => more parameters => Smaller error for each

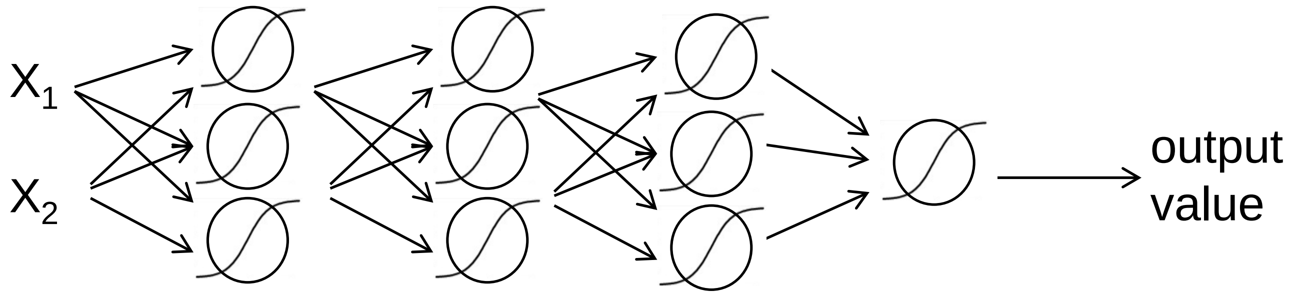
***especially* at lower layers**

Need:

More data and computing power (gpu)

Train with Care

More hidden layers => More varied features and transformations



But:

More layers => more parameters => Smaller error for each
***especially* at lower layers**

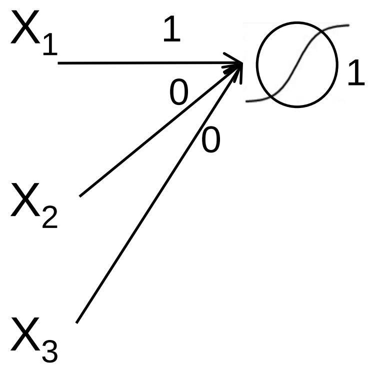
Need:

More data and computing power (gpu), functions that don’t saturate(RELU)

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)



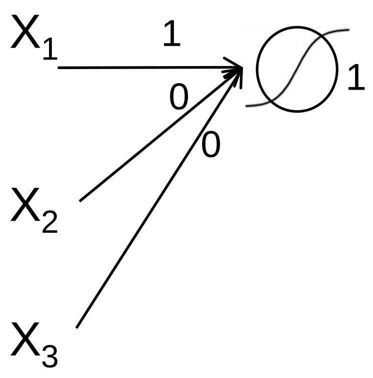
Call the connection parameters 'weights'.

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

*What feature transformation W*X 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’.

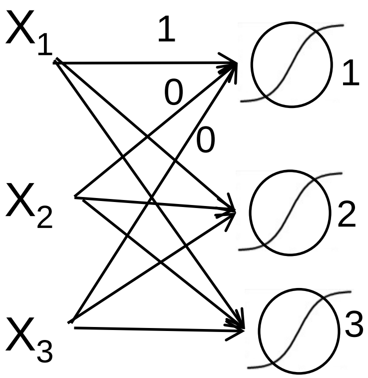
For node 1 let $[w_1\ w_2\ w_3] = [1\ 0\ 0]$

*What feature transformation W*X 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]

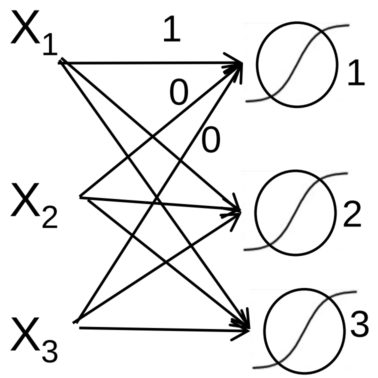
For node 2 let [w_1 w_2 w_3] = [0 1 0]

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

*What feature transformation W*X are these together?*

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 $[w_1\ w_2\ w_3\ ] = [0\ 1\ 0]$

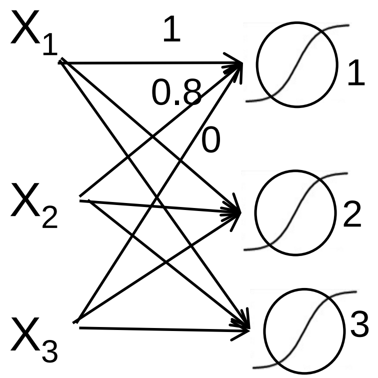
For node 3 let $[w_1\ w_2\ w_3\ ] = [0\ 0\ 1]$

*What feature transformation W*X 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 $[w_1\ w_2\ w_3] = [1 \quad 0.8 \quad 0]$

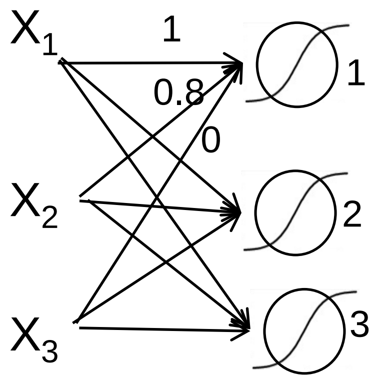
For node 2 let $[w_1\ w_2\ w_3] = [-0.8 \quad 1 \quad 0]$

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

*What feature transformation W*X 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?

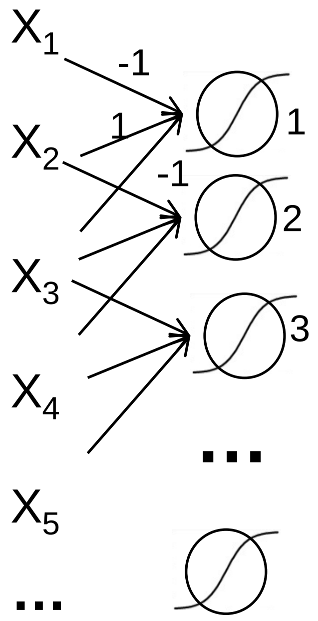
Informally, like projection onto 2 orthogonal dimensions (recall PCA example on Athletes Height and Weight!)

A Filter

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

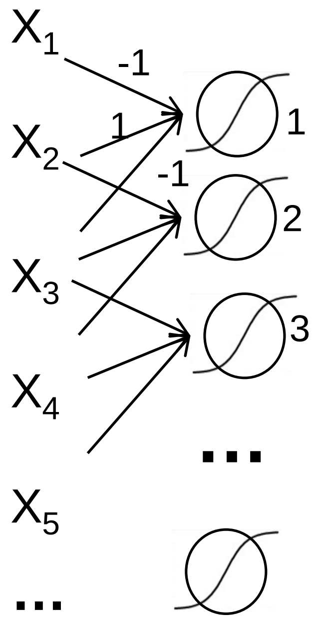
For node 1 let W= $[w_1\ w_2\ w_3\ ] = [-1\ 1\ -1]$

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)



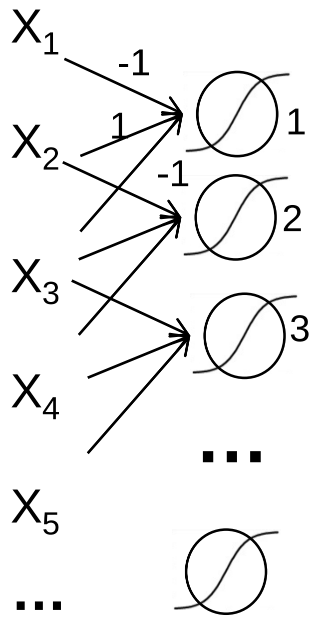
For node 1 let W= [w_1 w_2 w_3] = [-1 1 -1]

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)



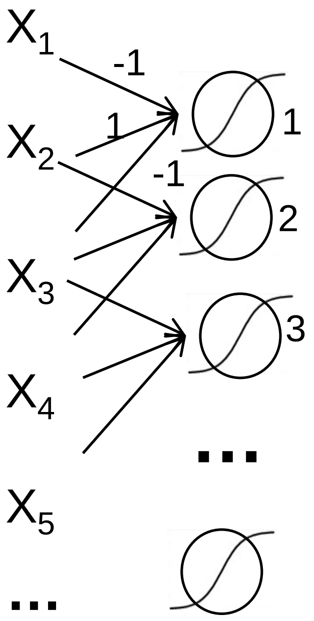
For node 1 let W= [$w_1\ w_2\ w_3$] = [-1 1 -1]

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

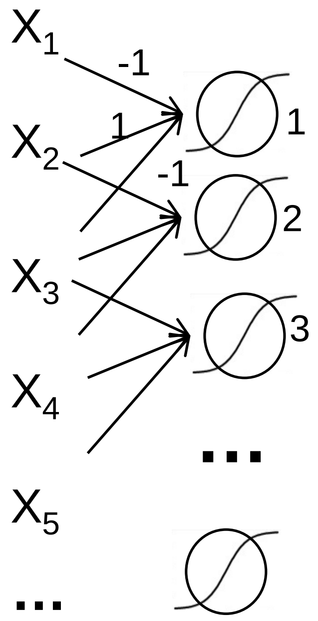
What is the hidden layer doing?

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?

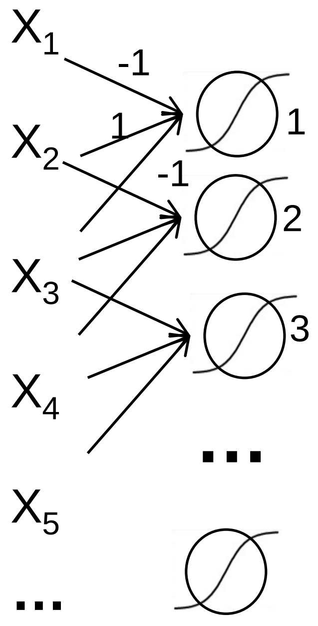
Informally, looking for a spike everywhere.

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.

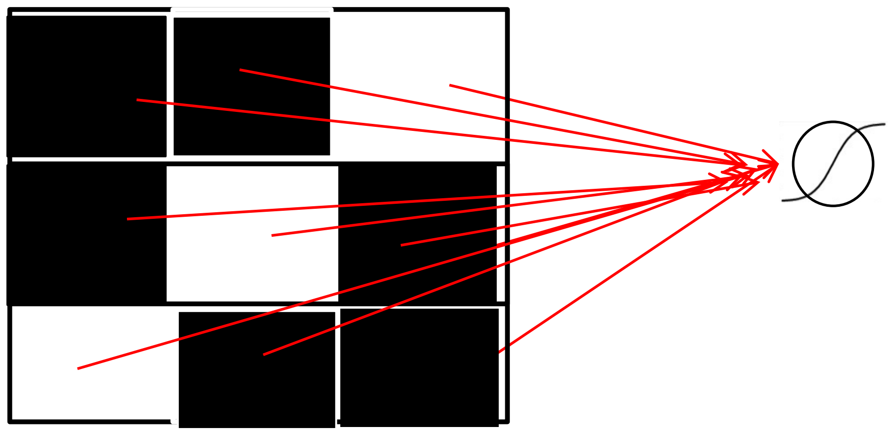
This is essentially a convolution operator, where W is the kernel.

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

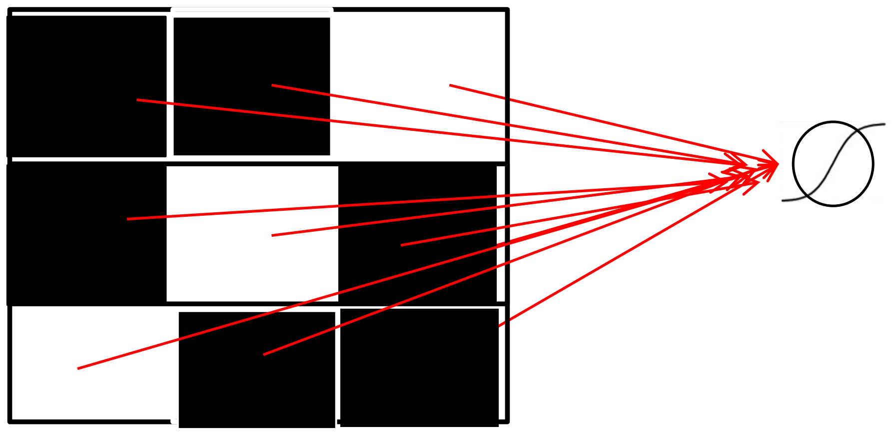
Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node



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)

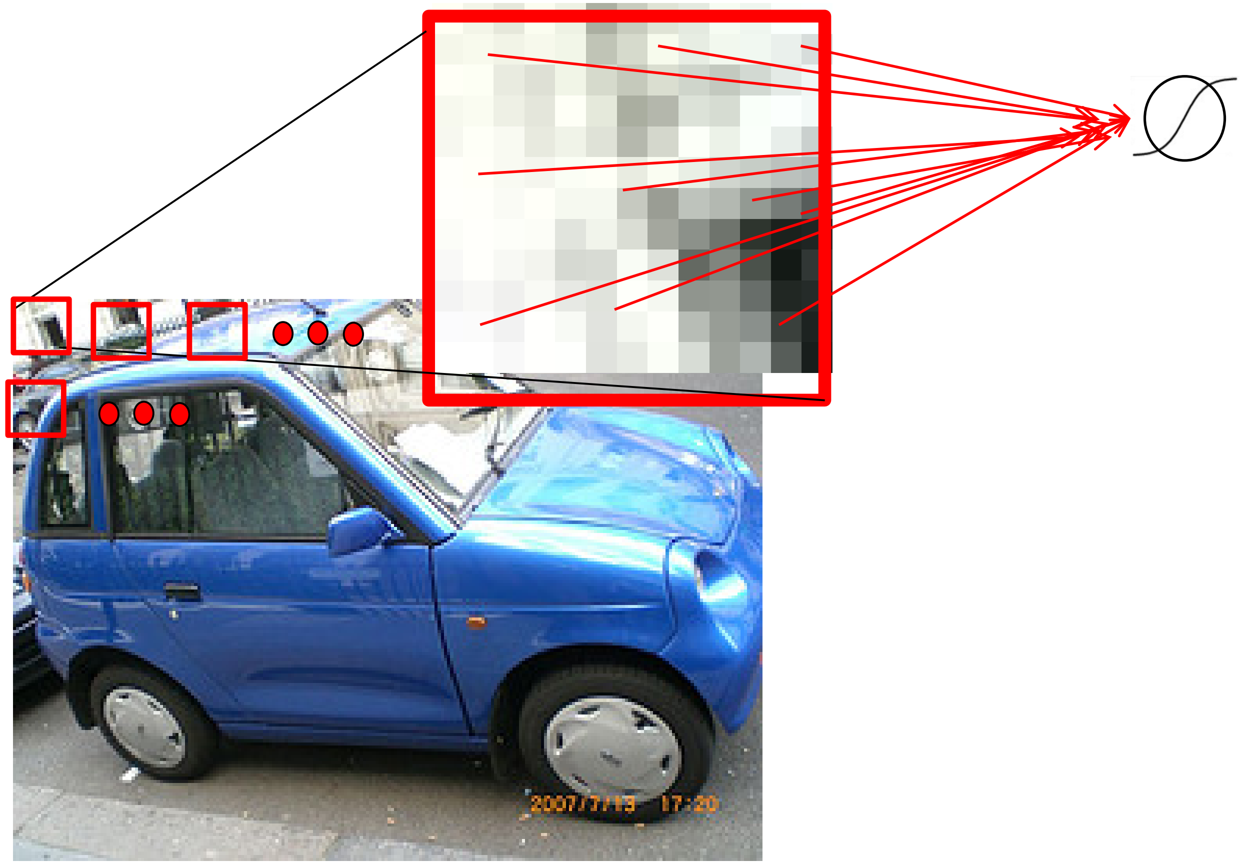
What W matrix would 'activate' for a upward-toward-left diagonal line?

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:

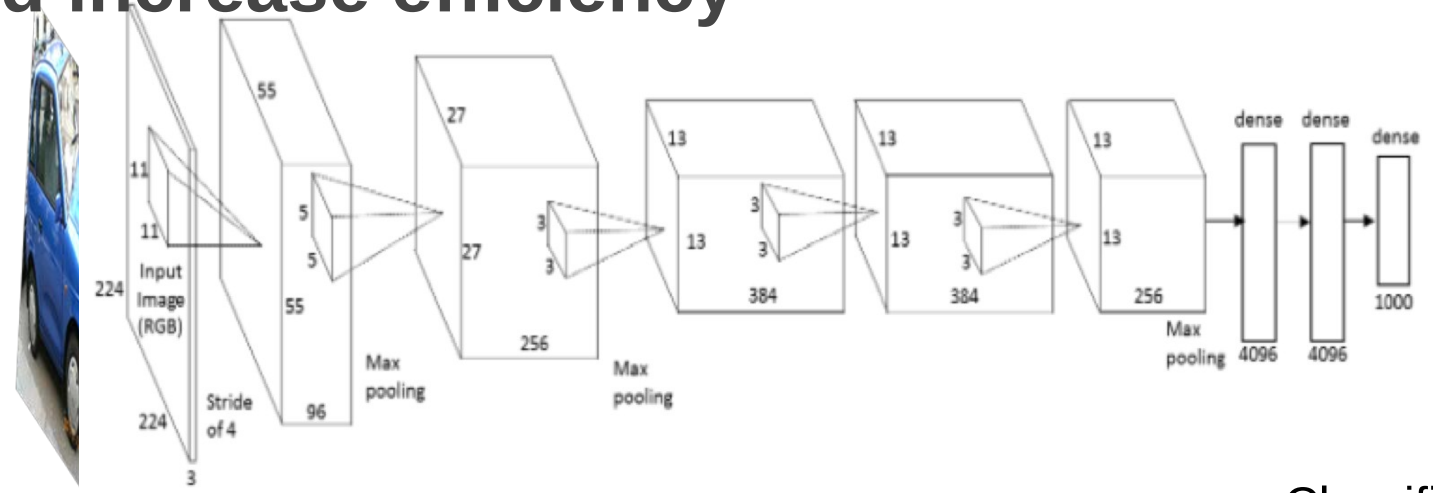
- filter size depends on input:
  - smaller filters for smaller details
  - 2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
  - smaller better but less efficient
- number of filters
  - depends on task
  - each filter is a new 2D layer

Convolution Network :

many layers and architecture options

Large Scale Versions

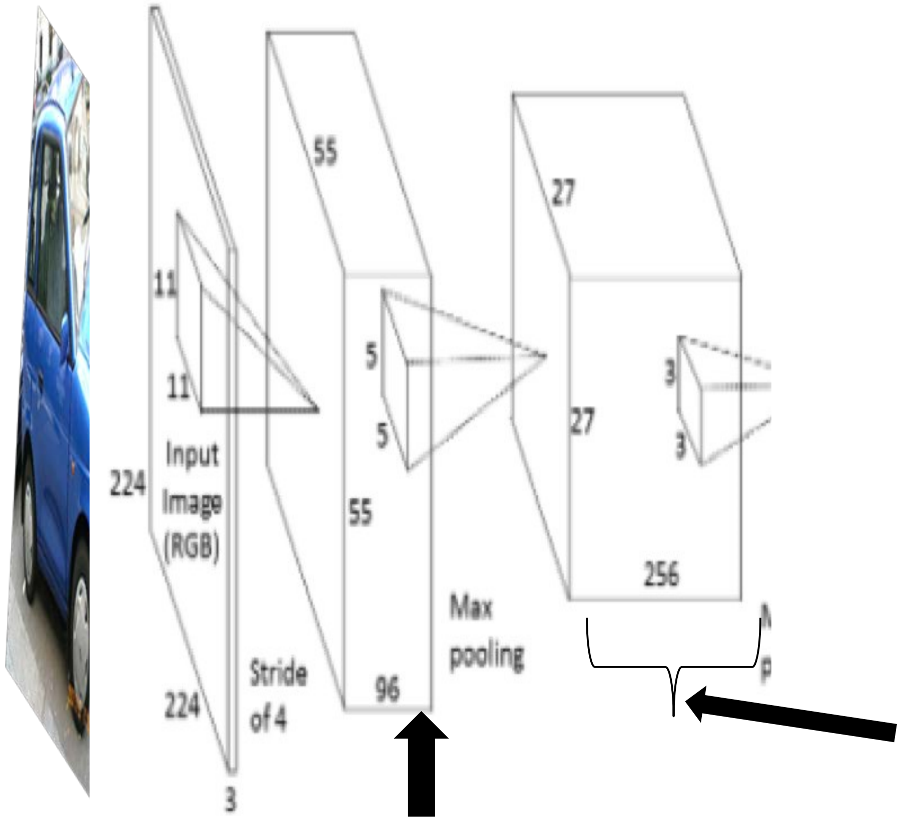
- **Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)**
- **Need large amounts of data and many heuristics to avoid overfitting and increase efficiency**



Convolution layers ➡ Classification layers and output

Large Scale Versions

- **Zooming in: Convolution layers**



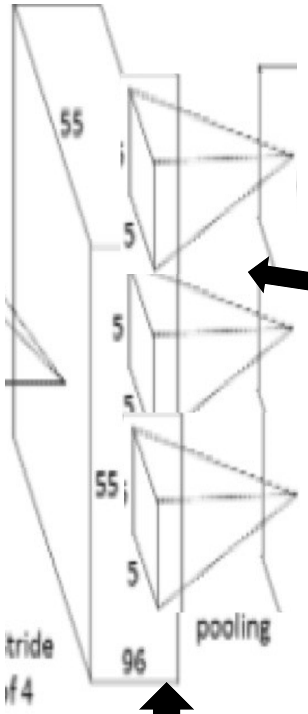
The thickness is the number of different convolutions, i.e. different transformations, sometimes called “channels”

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:**

Max pooling



Max pooling: take the maximum over a region and slide the region.

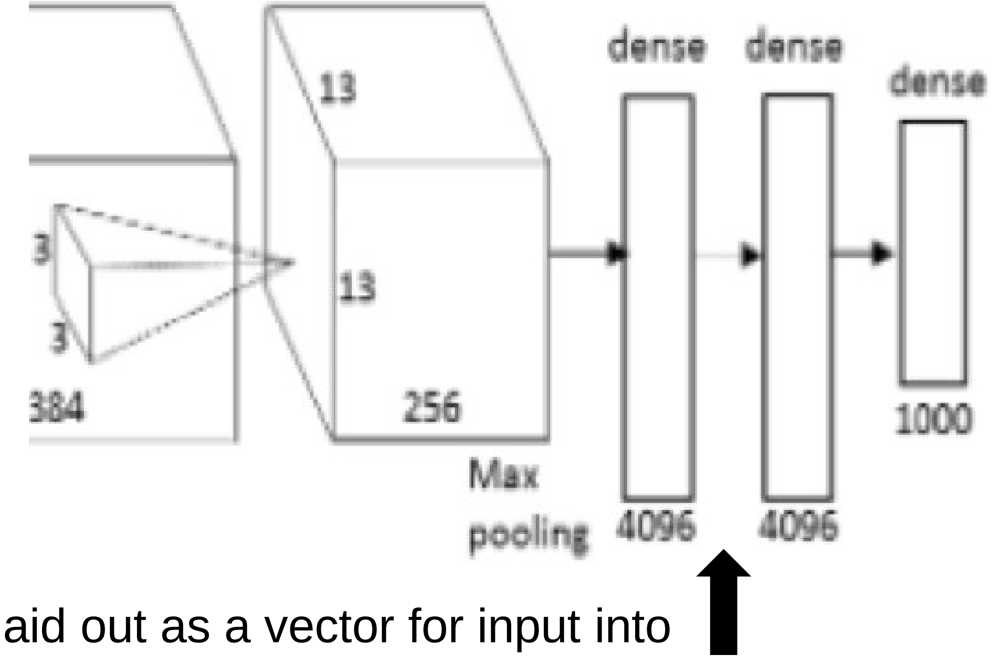
The larger the slide, the more down sampling occurs (which helps compute time)

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

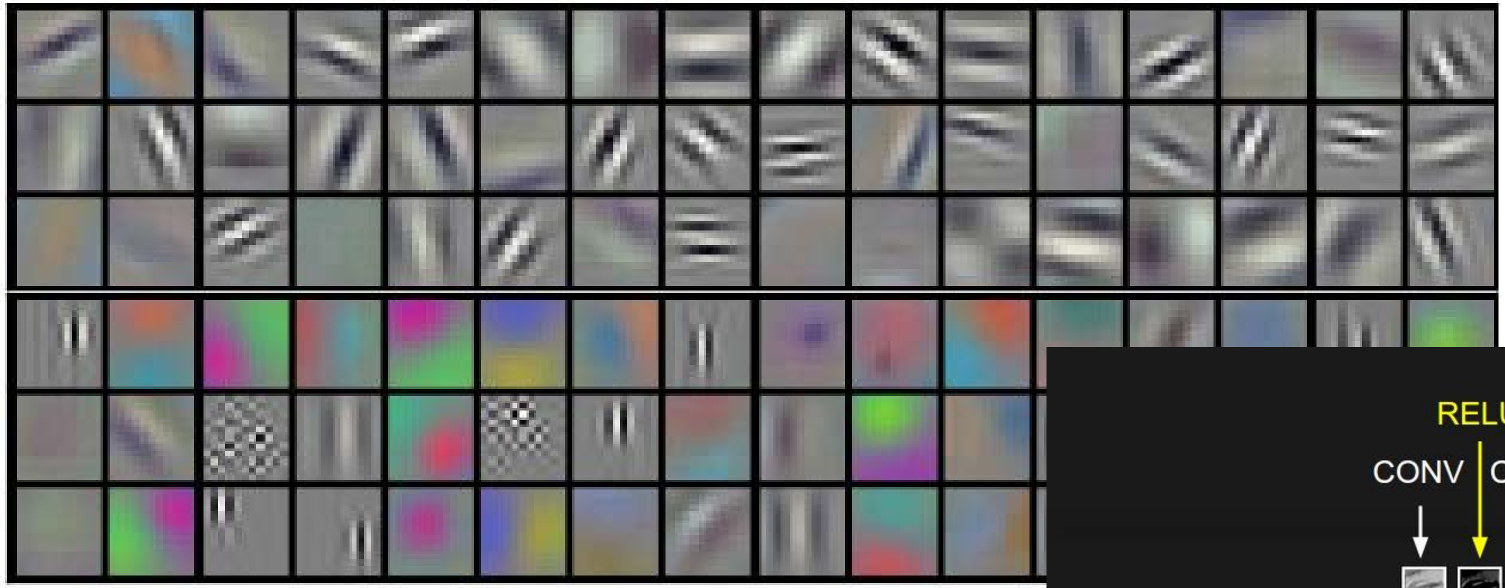
- **Zooming in:**

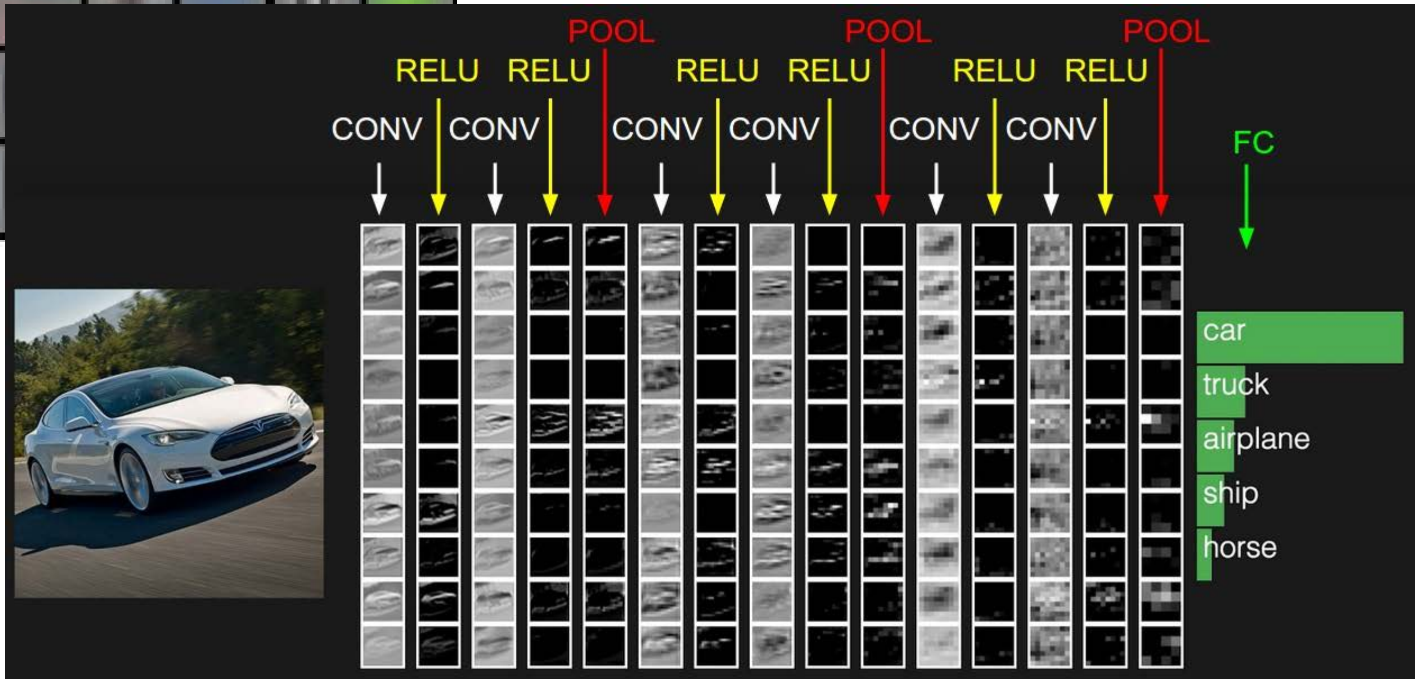
Classification layers



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





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

- Edge detection with Support Vector Machine

  OR

  Convolution Neural Network?

- With small datasets and reasonable features, SVMs can work well
- But building features is hard, and large classification problems can benefit from common features, and CNNs discover features

The Zoo

Machine learning/convolution network frameworks:

tensorflow, pyTorch, Theano

(libraries and API to build graphs of networks and processing)

Caffe – C/C++ library with many pretrained models

Keras - higher level CNN library with tensorflow or Theano

Darknet – C++ library, object detection

Matlab – CNN functions, and pretrained networks

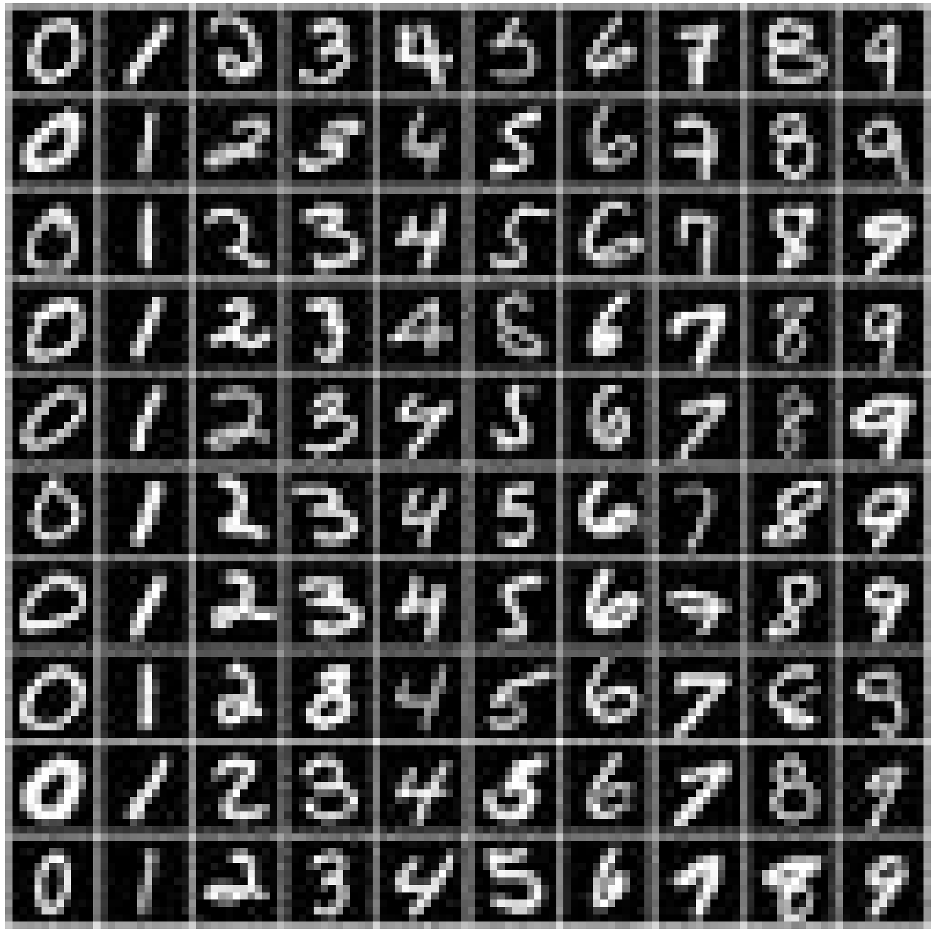
Many others, many network implementations

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**



MNIST on Comet

- **Login to comet**

- **Get an interactive compute node session**
  - getcpu
- **Start up singularity image:**
  - module load singularity
  - singularity shell /home/train129/keras-tensorflow-cpu.img
  - jupyter notebook --no-browser --ip="*" &

Cut and paste http address, edit localhost, look in DeepLearningTutorial for notebook

LabMNIST_Final Last Checkpoint: 7 minutes ago (autosaved)

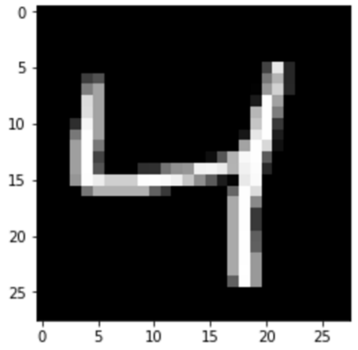
```
for i in range(0,3):
   im = Image.fromarray(X_train[i,:,:])
   im.save("Xtrain_num"+str(i)+"_cat_"+str(Y_train[i])+".jpeg")

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

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

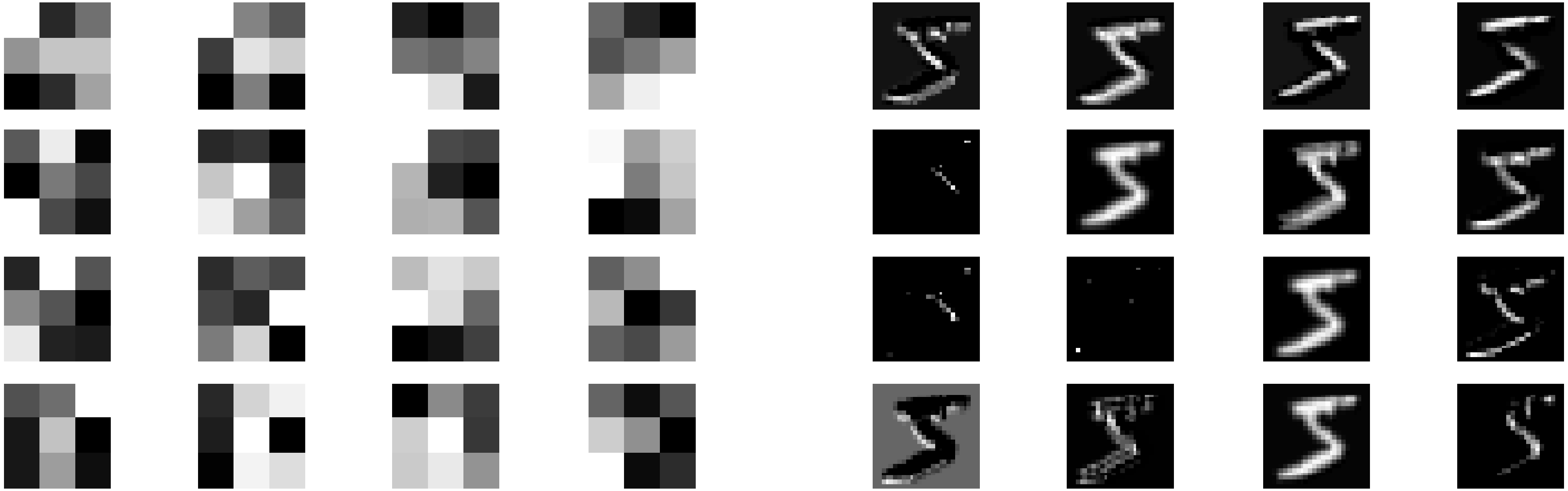
```
(5000, 28, 28)

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

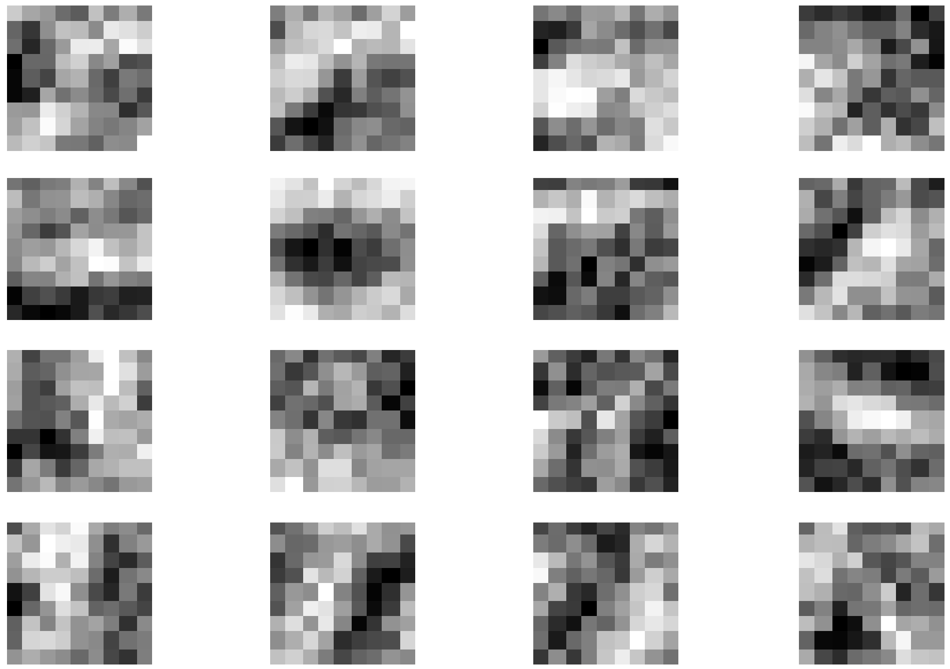


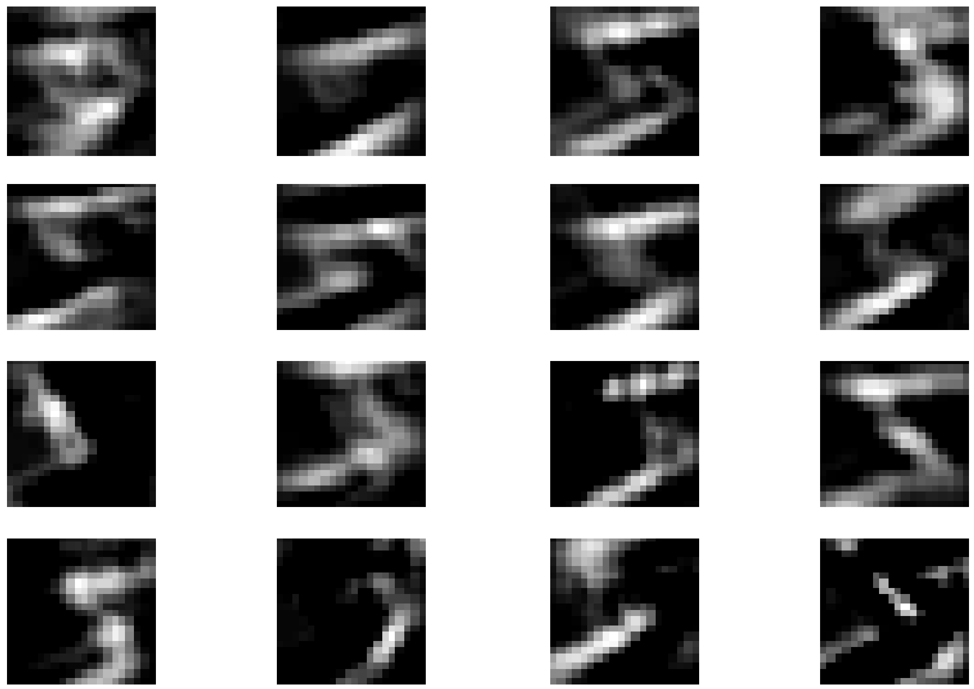

```
img load done
```

3x3 first convolution layer filter and activation



9x9 first convolution layer filter and activation





Pause