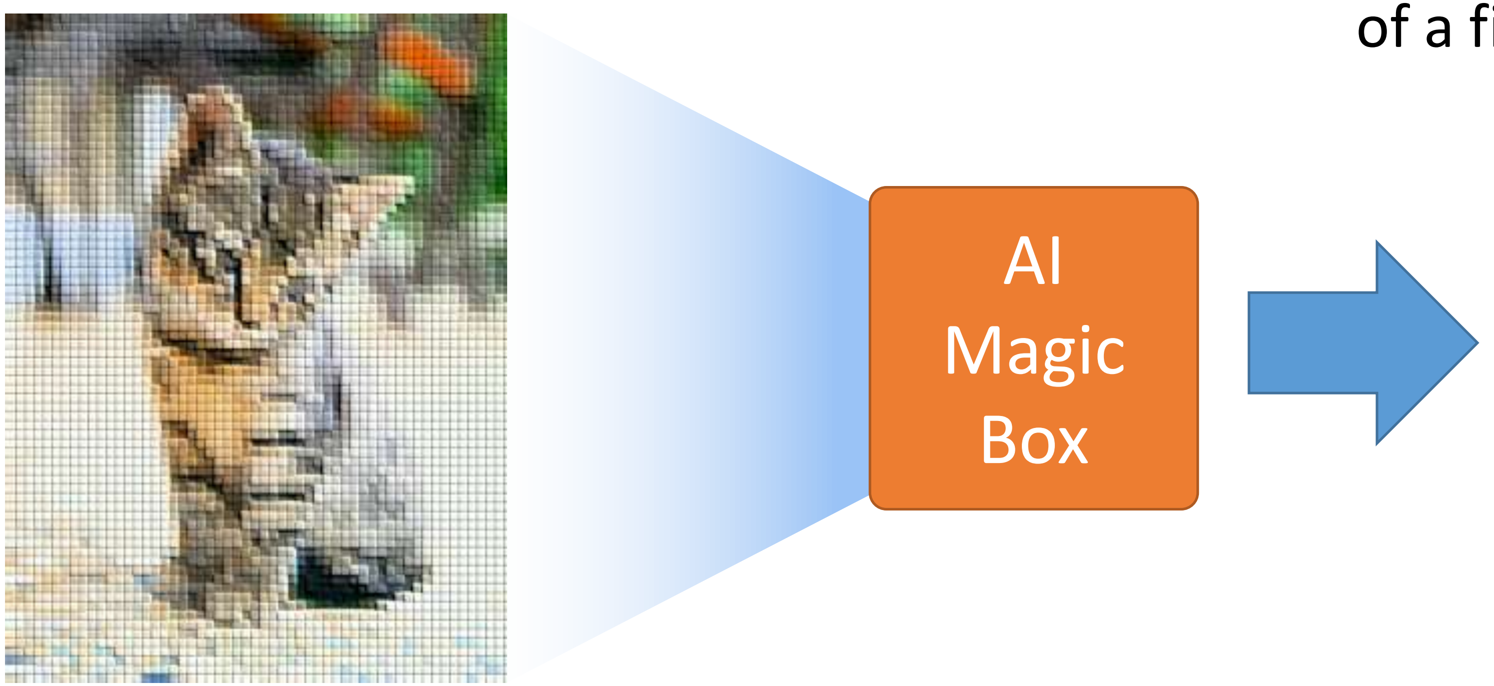
Linear Classifiers

Notes based on
EECS 498-007 / 598-005
Deep Learning for Computer Vision
At University of Michigan

Previously: Image Recognition

Input: image

Output: Assign image to one of a fixed set of categories



cat
bird
deer
dog
truck

All images from https://unsplash.com/s/photos/cat

Previously: Challenges of Recognition

Viewpoints

Variation within class

Background blending

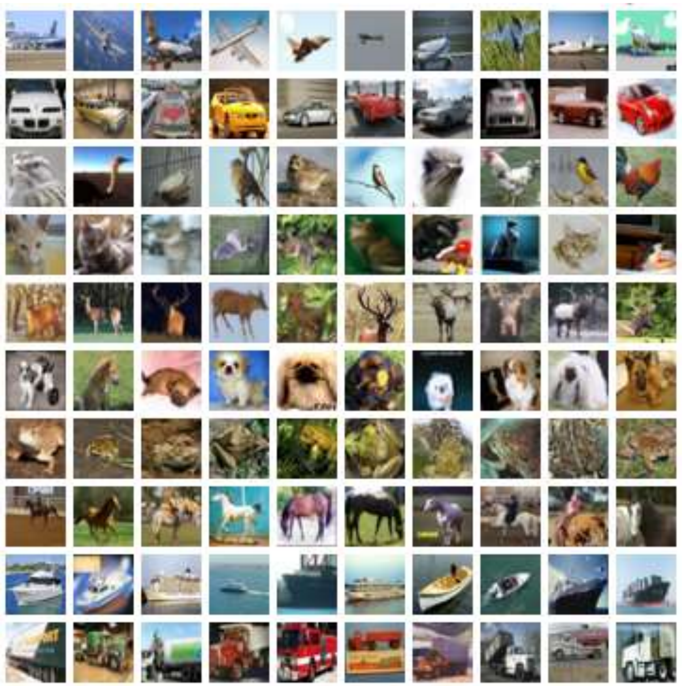
Illumination

Deformation

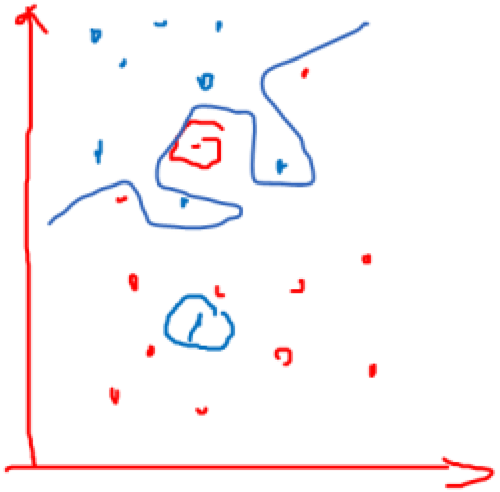
Occlusion

Previously: Data-driven approach, k-NN

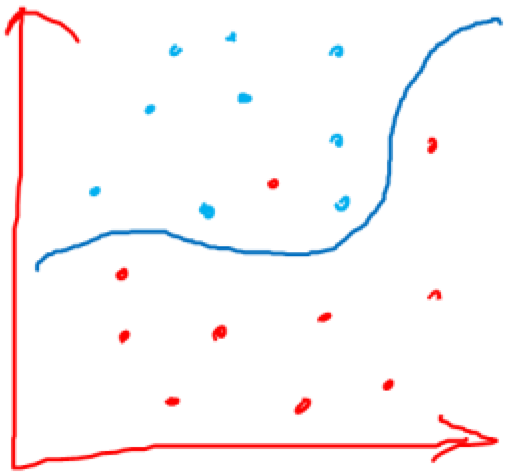
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



1-NN classifier



3-NN classifier



train | test

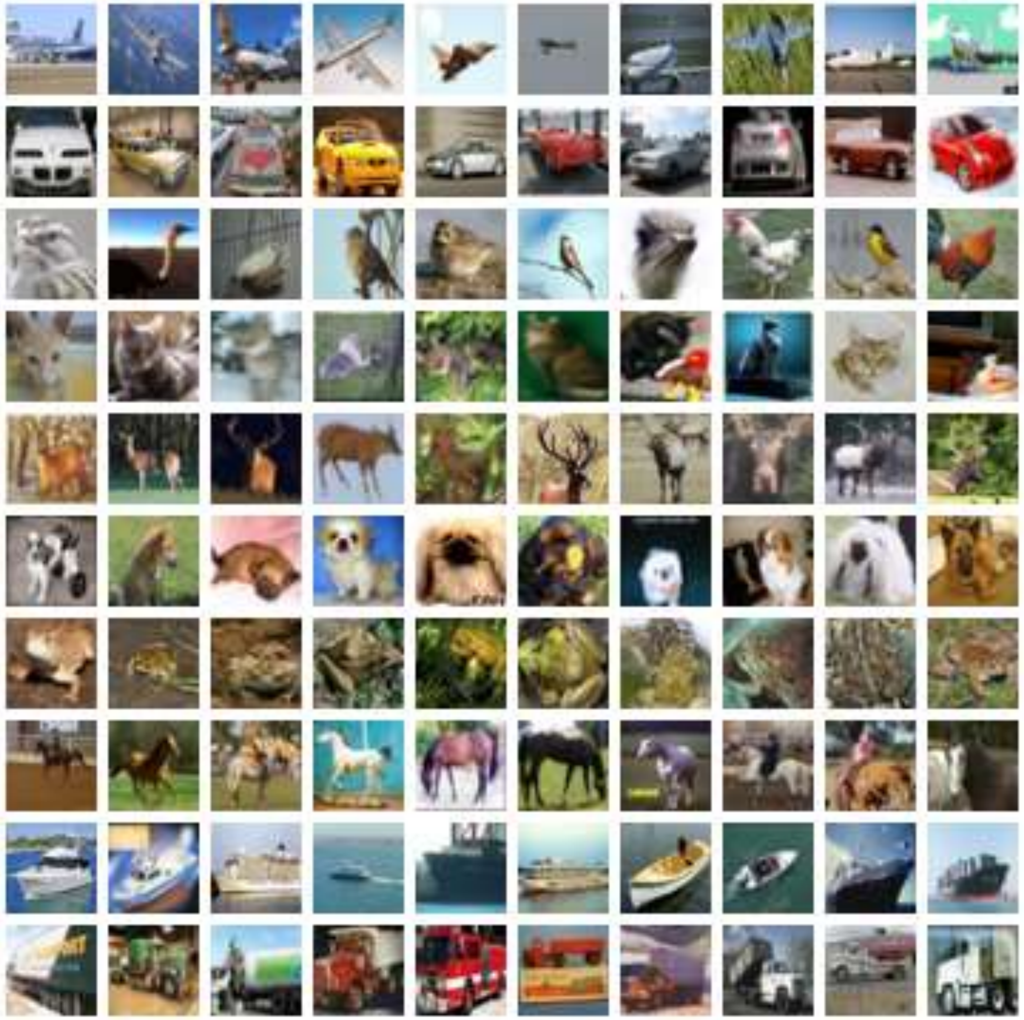
train | validate | test

Today: Linear Classifiers

Notes based on
EECS 498-007 / 598-005
Deep Learning for Computer Vision
At University of Michigan

Recall CIFAR10

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



50,000 training images
Each image is **32x32x3**

10,000 test images.

Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

$$f(\mathbf{x}, \mathbf{W})$$

10 numbers giving
class scores

$\mathbf{W}$

parameters
or weights

Parametric Approach: Linear Classifer

$$f(x, W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(x, W)$

W

parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifer

Image

Array of **32x32x3** numbers
(3072 numbers total)

(3072,1)

$$\underbrace{\boxed{f(x, W)}}_{(10,\,1)} = \underbrace{\boxed{W}}_{(10,\,3072)}\boxed{x}$$

$f(x, W)$

10 numbers giving
class <u>scores</u>

W

parameters
or weights

Parametric Approach: Linear Classifer

$$\underbrace{\boxed{f(x, W)}}_{(10,\ 1)} = \underbrace{\boxed{W}}_{(10,\ 3072)}\overbrace{\boxed{x}}^{(3072,1)} + \boxed{b} \quad (10,1)$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow f(x, W) \longrightarrow$ 10 numbers giving class <u>scores</u>

$\uparrow$

W

parameters
or weights

Parametric Approach: Linear Classifer

(3072,1)

$$\boxed{f(x, W)} = \boxed{W}\boxed{x} + \boxed{b}$$

(10,1)

(10, 1) (10, 3072)

(classes, 1) (classes, inputPoints)

Image

Array of **32x32x3** numbers
(3072 numbers total)

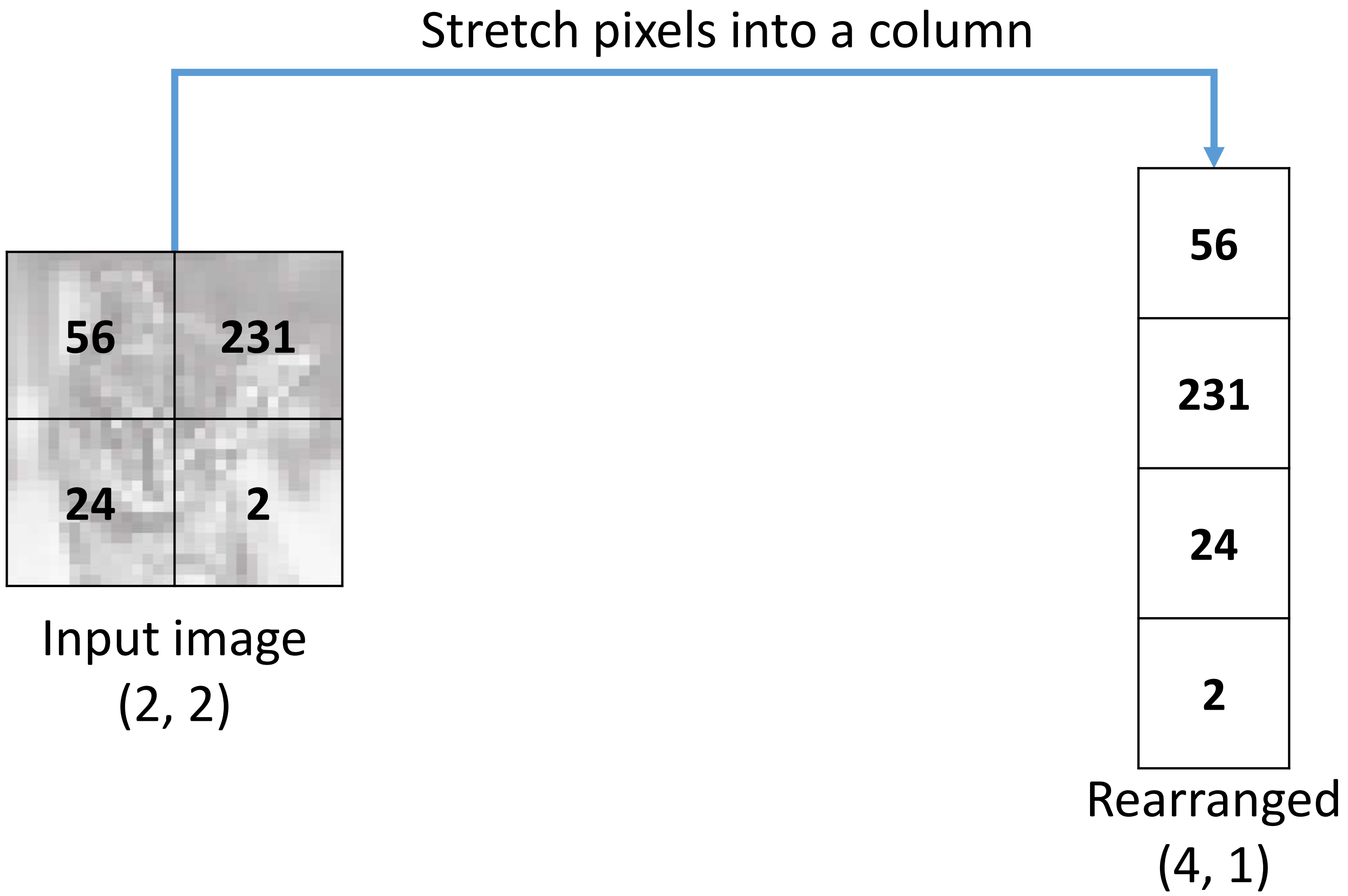
$f(x, W)$

10 numbers giving
class <u>scores</u>

W
parameters
or weights

Example for 2x2 image, 3 classes (cat / dog / ship)

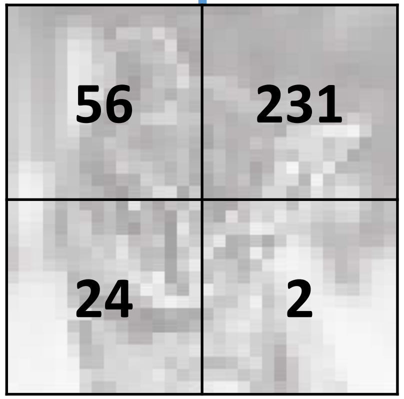
$$f(x, W) = Wx + b$$



Example for 2x2 image, 3 classes (cat / dog / ship)

Stretch pixels into a column

$$f(x, W) = Wx + b$$



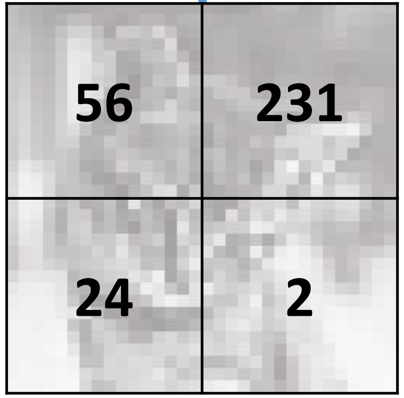
Input image
(2, 2)

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W\ (3,\,4)} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{\text{Input}\ (4,\,1)} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{\textit{bias}\ (3,\,1)} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\text{Scores}\ (3,\,1)} \begin{array}{l} \text{cat} \\ \text{dog} \\ \text{ship} \end{array}$$

Linear Classifier: Algebraic Viewpoint

Stretch pixels into a column

$$f(x, W) = \color{red}{W}\color{blue}{x} + b$$



Input image
(2, 2)

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{\color{red}{W}\ \color{red}{(3,\,4)}} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{\color{blue}{\text{Input}}\ \color{blue}{(4,\,1)}} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{\textit{bias}\ (3,\,1)} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\text{Scores}\ (3,\,1)} \begin{matrix} \text{cat} \\ \text{dog} \\ \text{ship} \end{matrix}$$

Linear Classifier: Predictions are **Linear**

$$f\,(x,\ W) = Wx$$ (ignore bias)

$$f\,(cx,\ W) = W(cx) = c * f\,(x,\ W)$$

Linear Classifier: Predictions are **Linear**

$$f(x, W) = Wx$$ (ignore bias)

$$f(cx, W) = W(cx) = c * f(x, W)$$

Image

Scores

-96.8
437.9
61.95

0.5 * Image

0.5 * Scores

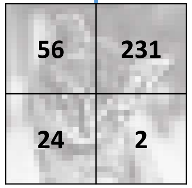
-48.4
218.9
31.0

Interpreting a Linear Classifier

Algebraic Viewpoint

$$f(x, W) = Wx + b$$

Stretch pixels into a column



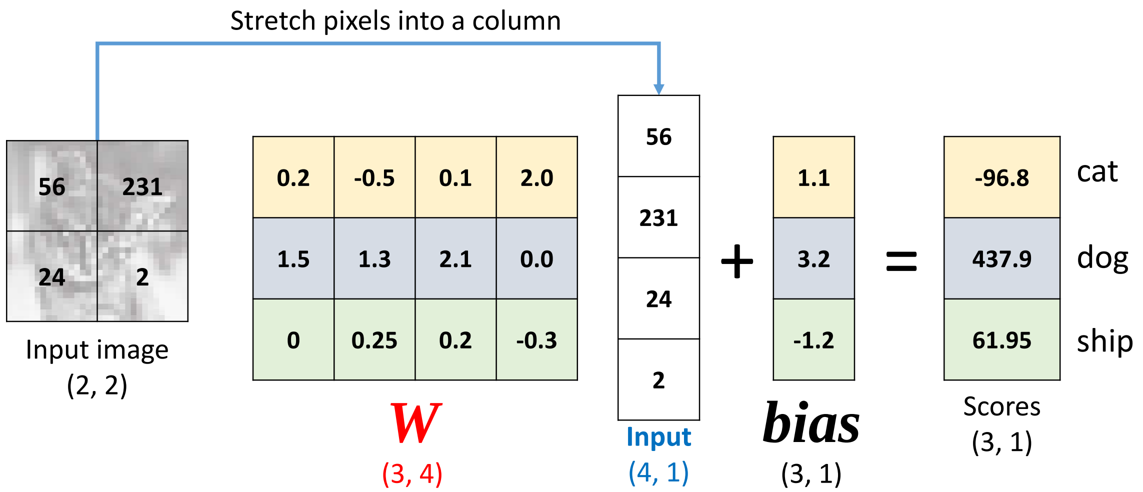
Input image
(2, 2)

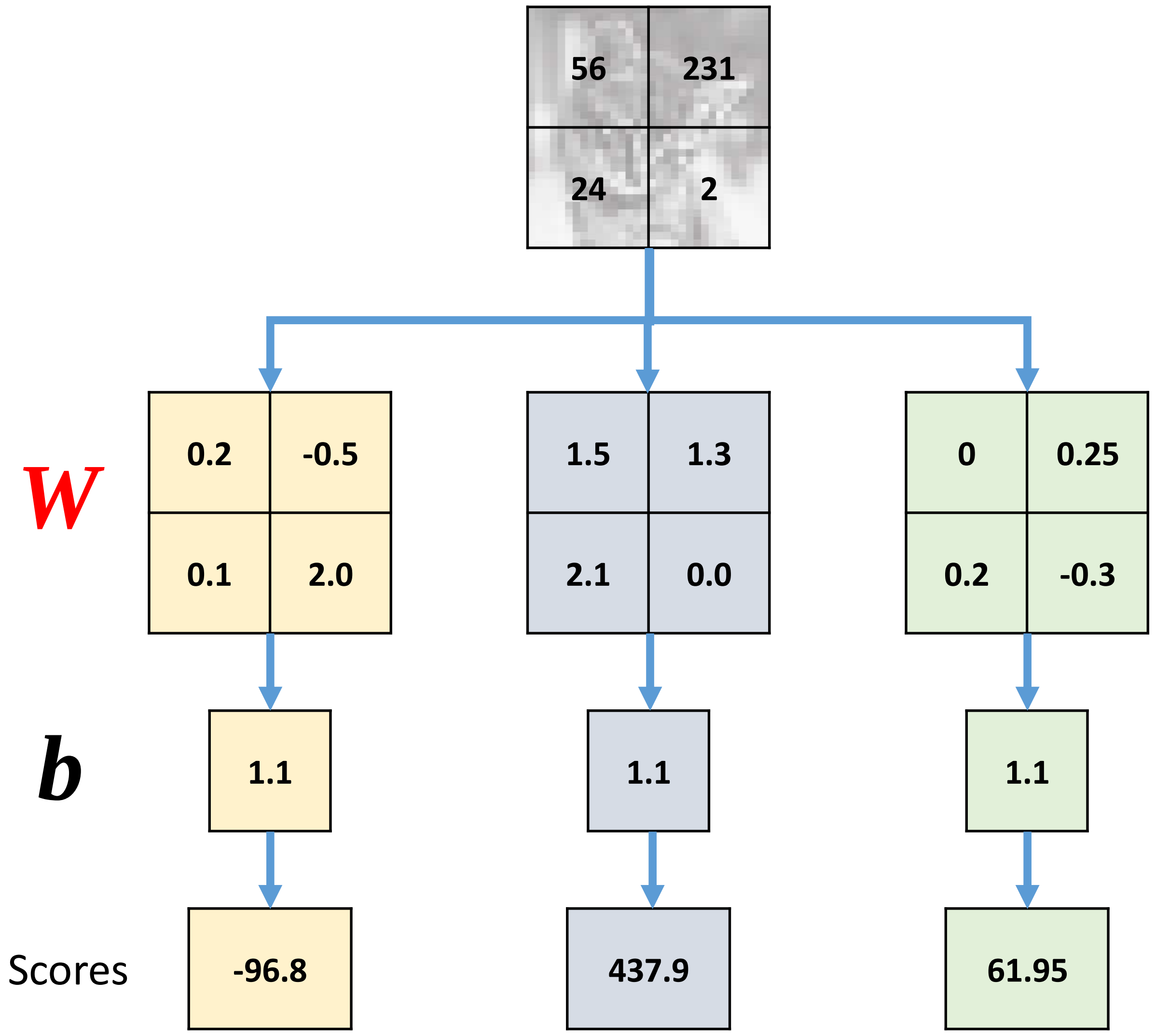
$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{\substack{W \\ (3,\ 4)}} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{\substack{\text{Input} \\ (4,\ 1)}} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{\substack{bias \\ (3,\ 1)}} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\substack{\text{Scores} \\ (3,\ 1)}} \begin{array}{l} \text{cat} \\ \text{dog} \\ \text{ship} \end{array}$$

Interpreting a Linear Classifier

Algebraic Viewpoint

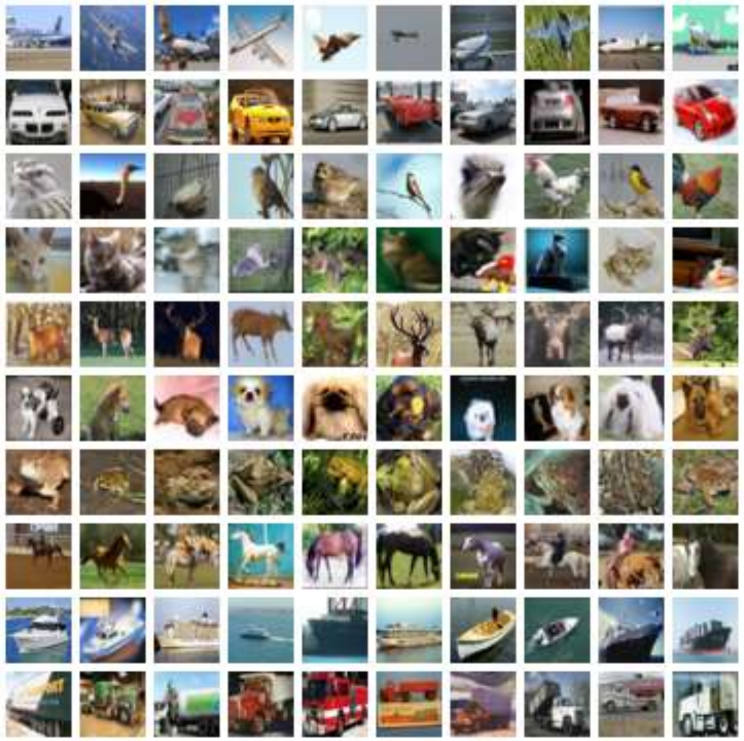
$$f(x, W) = Wx + b$$

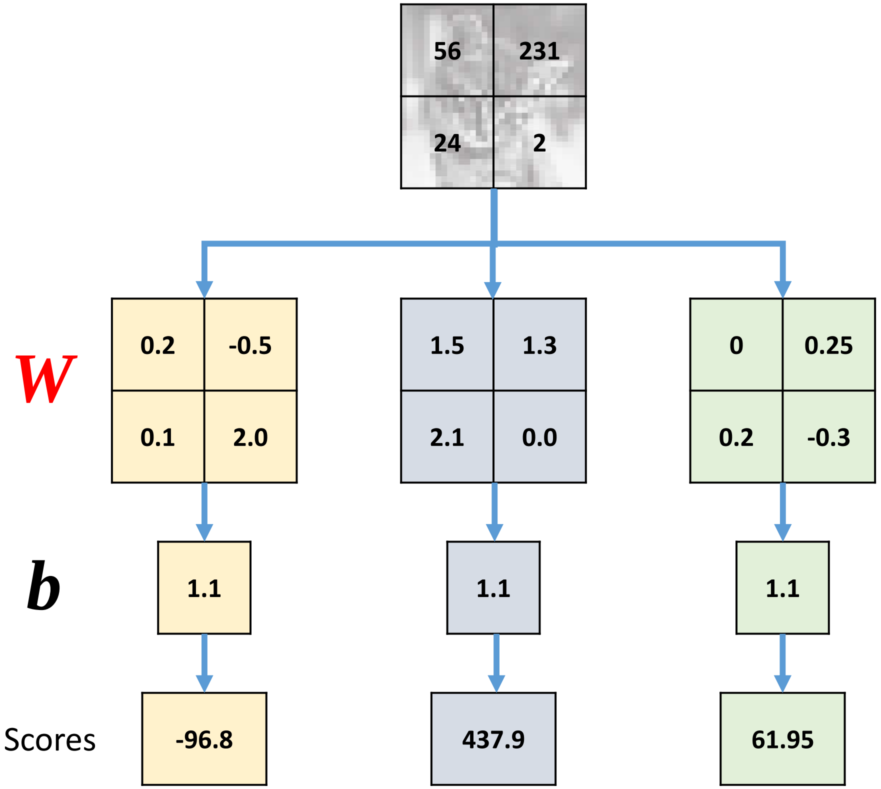




Interpreting a Linear Classifier

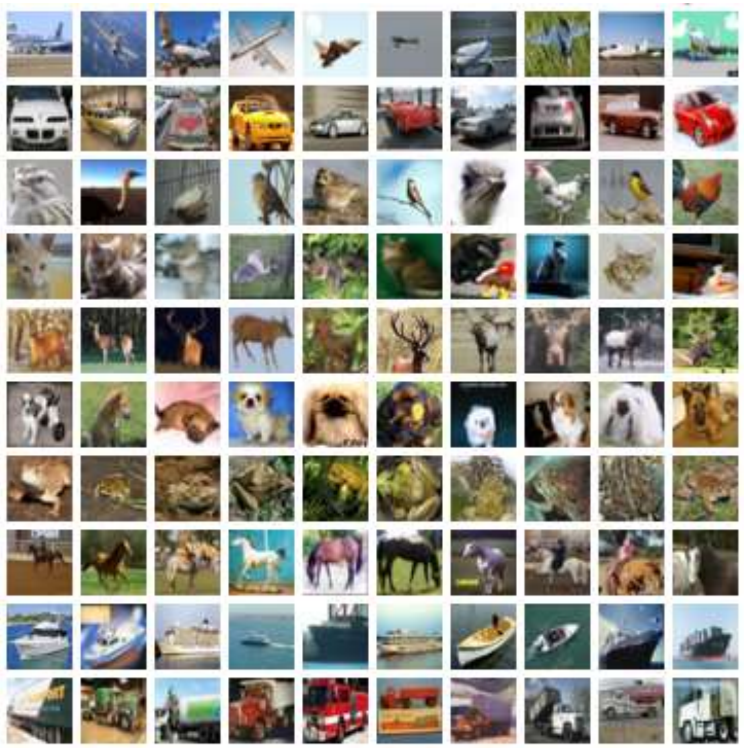
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

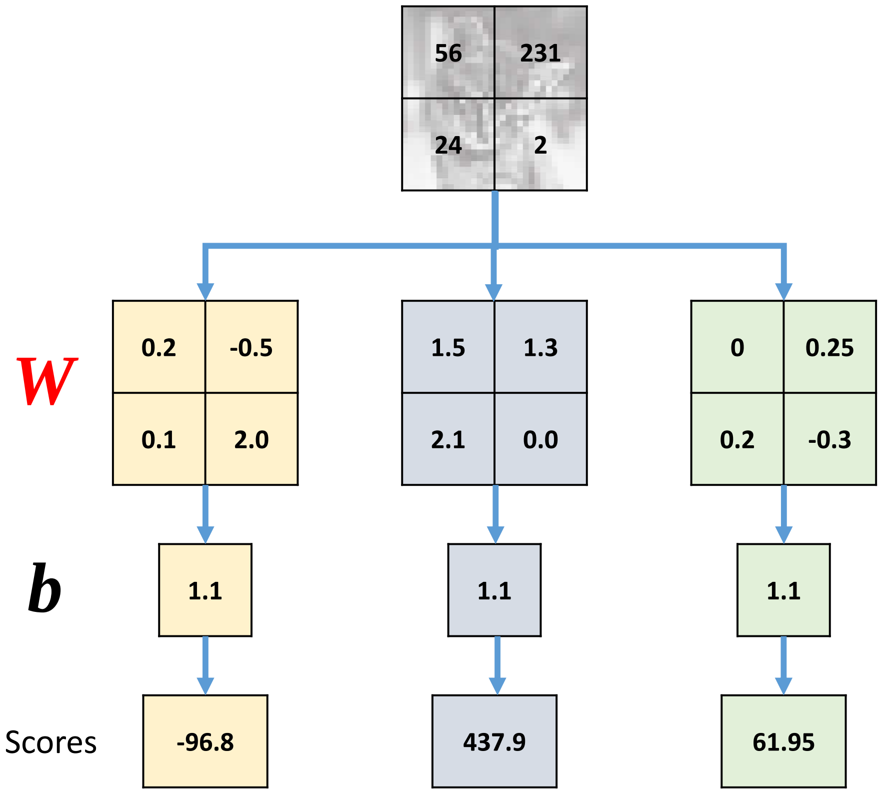




Interpreting a Linear Classifier: Visual Viewpoint

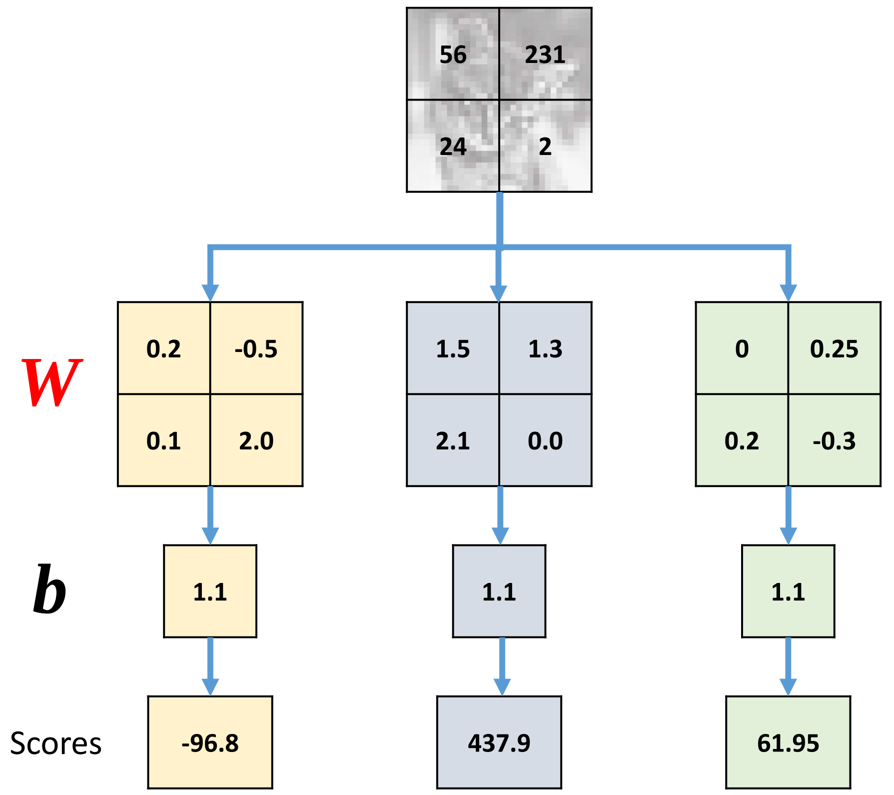
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck





Interpreting a Linear Classifier: Visual Viewpoint

Linear classifier has one "template" per category

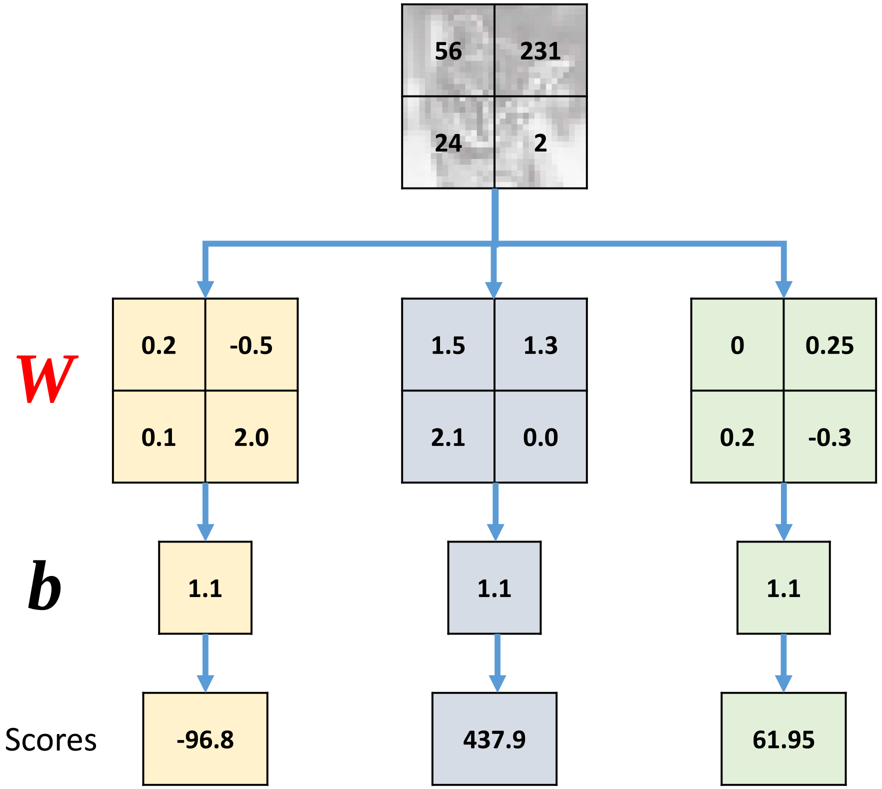


Interpreting a Linear Classifier: Visual Viewpoint

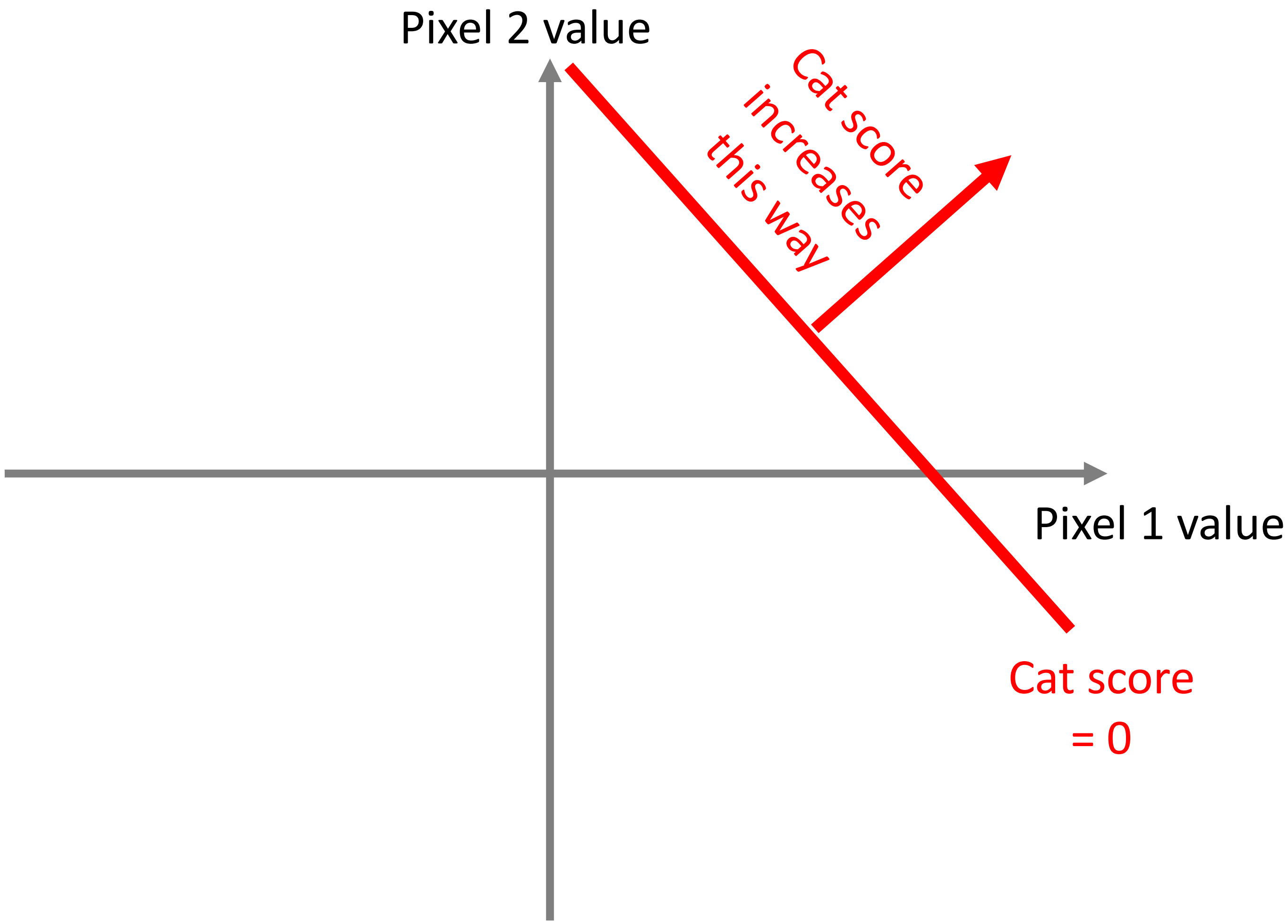
Linear classifier has one "template" per category

A single template cannot capture multiple modes of the data

Example: horse template has two heads!



Interpreting a Linear Classifier: Geometric Viewpoint

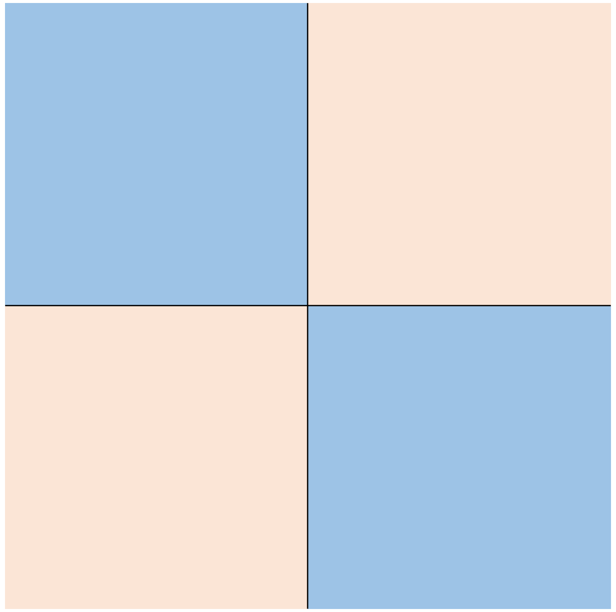


$$f(x, W) = Wx + b$$

Hard Cases for a Linear Classifier

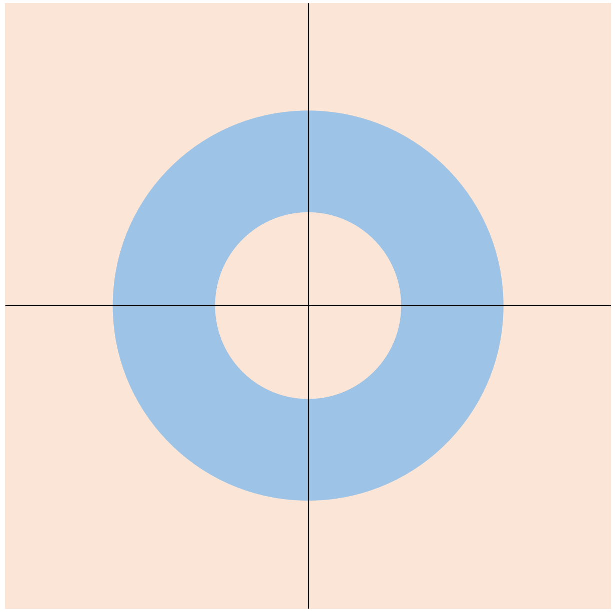
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



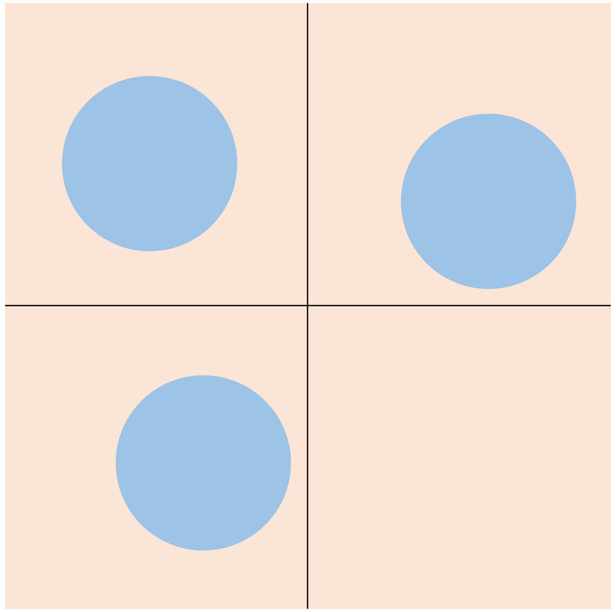
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



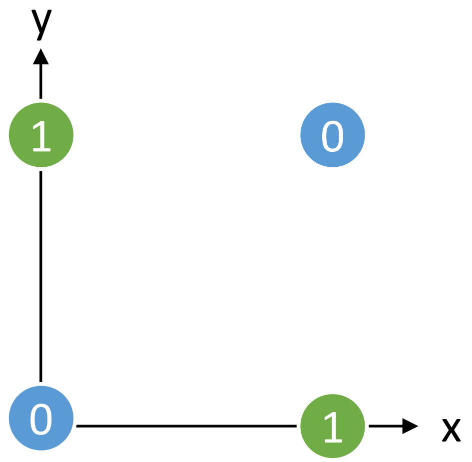
Class 1:
Three modes

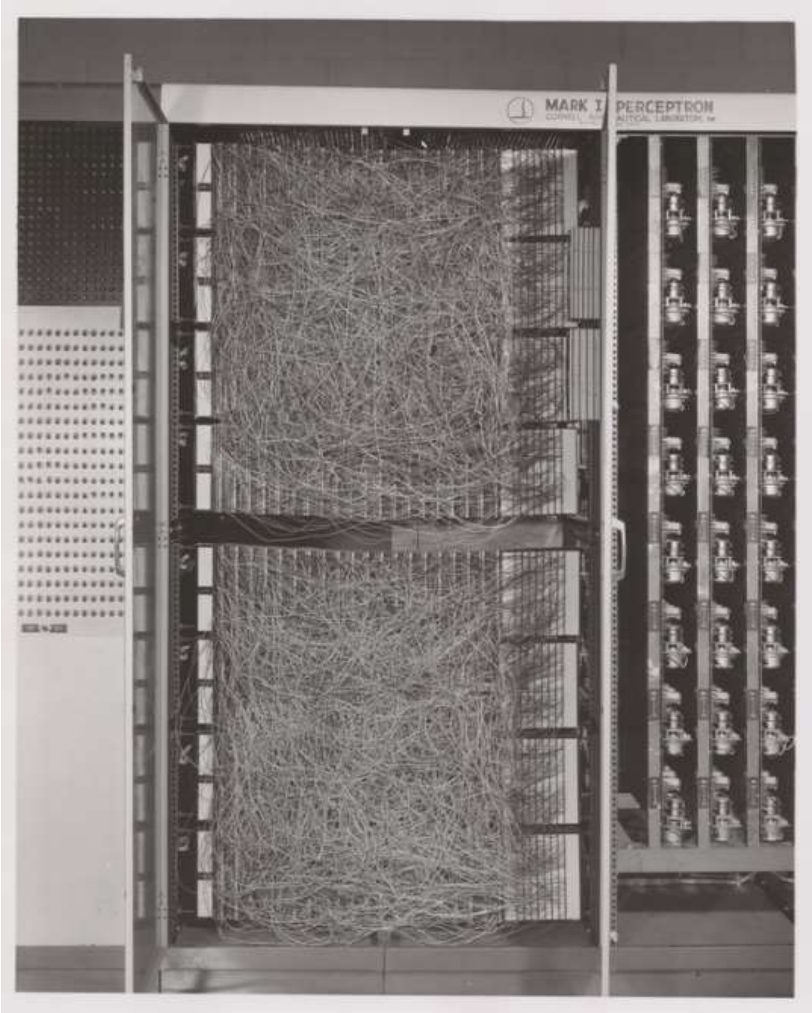
Class 2:
Everything else



Recall: Perceptron couldn't learn XOR

X	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0





So Far: Defined a linear score function

$$f(x, W) = Wx + b$$

airplane	-3.45	-0.51	3.42
automobile	-8.87	6.04	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	**3.58**	5.55
frog	3.78	4.49	-4.34
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	**-4.79**
truck	-0.72	-2.93	6.14

Given a W, we can compute class scores for an image x.

But how can we actually choose a good W ?

We need to choose good weights, ***W***

$f(x, W) = Wx + b$



airplane	-3.45	-0.51	3.42
automobile	-8.87	6.04	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	**3.58**	5.55
frog	3.78	4.49	-4.34
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	**-4.79**
truck	-0.72	-2.93	6.14

TODO:

1. Use a **loss function** to quantify how good a value of W is

2. Find a W that minimizes the loss function (**optimization**)

Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier

High loss = bad classifier

(Also called: objective function; cost function)

Negative loss function sometimes called
reward function, profit function,
utility function, fitness function, etc

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and

y_i is label

Loss for a single example is

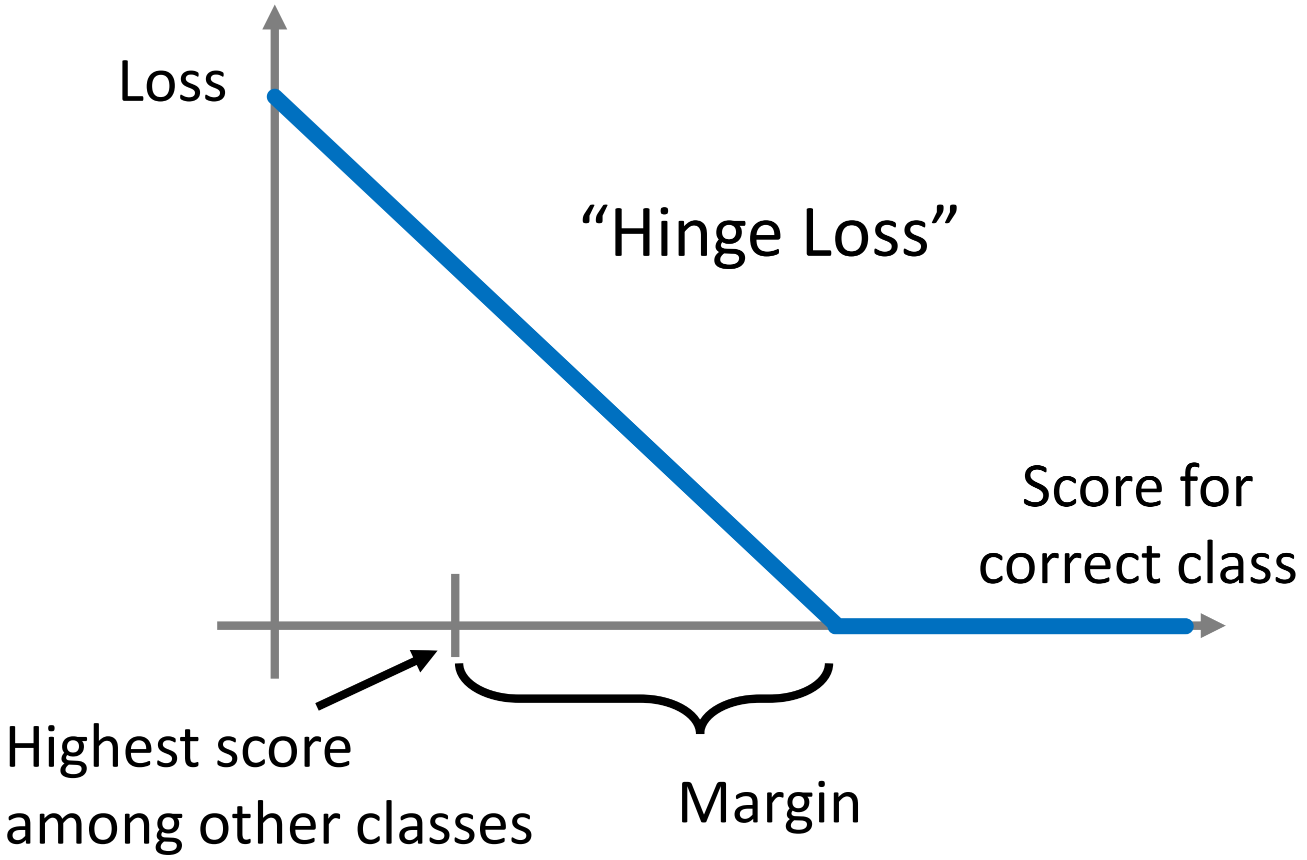
$$L_i(f(x_i, W), y_i)$$

Loss for the dataset is average of per-example losses:

$$L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)$$

Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"



Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM Loss

cat	**3.2**	1.3	2.2
dog	5.1	**4.9**	2.5
ship	-1.7	2.0	**-3.1**

Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM Loss

cat	**3.2**	1.3	2.2
dog	5.1	**4.9**	2.5
ship	-1.7	2.0	**-3.1**
Loss	2.9		

Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 5.1 - 3.2 + 1)
+ max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

Multiclass SVM Loss

cat	**3.2**	1.3	2.2
dog	5.1	**4.9**	2.5
ship	-1.7	2.0	**-3.1**
Loss	2.9	0	

Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 1.3 - 4.9 + 1)
+ max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

Multiclass SVM Loss

cat	**3.2**	1.3	2.2
dog	5.1	**4.9**	2.5
ship	-1.7	2.0	**-3.1**
Loss	2.9	0	12.9

Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 2.2 - (-3.1) + 1)
+ max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

Multiclass SVM Loss

			
cat	**3.2**	1.3	2.2
dog	5.1	**4.9**	2.5
ship	-1.7	2.0	**-3.1**
Loss	2.9	0	12.9

Given an example (x_i, y_i)
(x_i is image, y_i is label),

let $s = f(x_i, W)$ be scores,

then the SVM loss has the form:

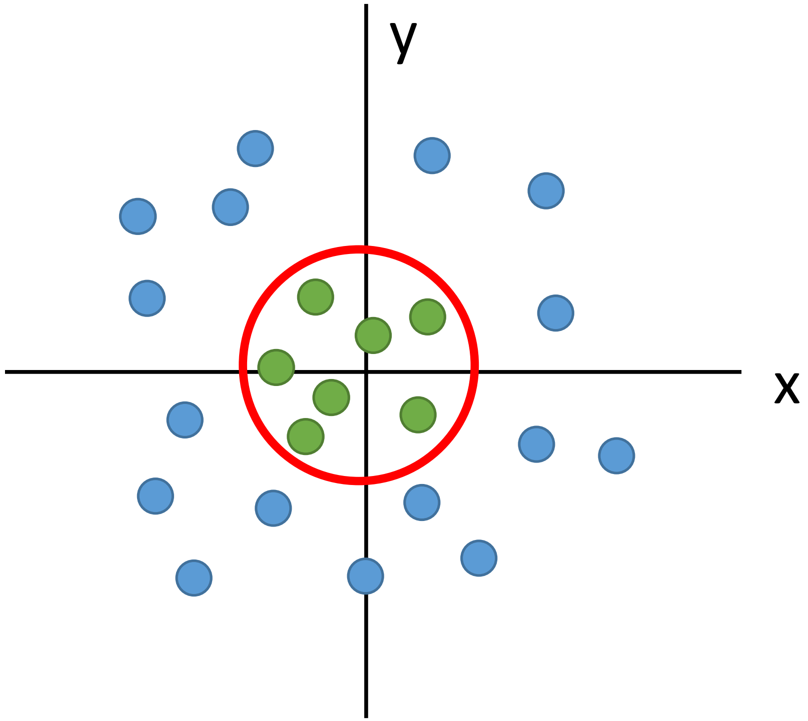
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Loss over the dataset is:

L = (2.9 + 0.0 + 12.9) / 3
= 5.27

Problem: Linear Classifiers aren't that powerful

Original space

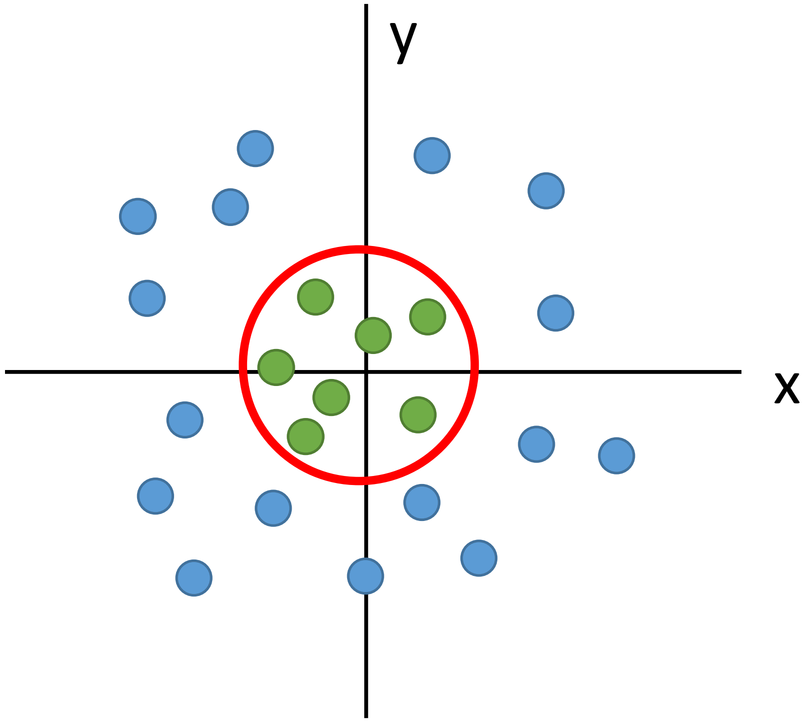


Visual Viewpoint

One template per class:
Can't recognize different
modes of a class

One solution: **Feature Transforms**

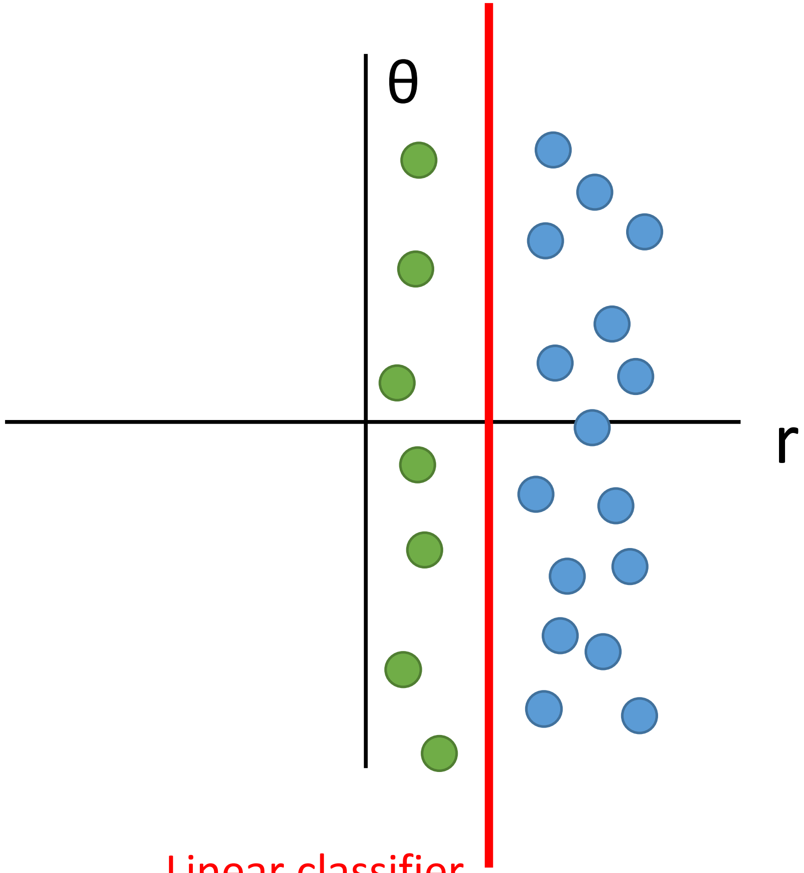
Original space



$$r = (x^2 + y^2)^{1/2}$$
$$\theta = \tan^{-1}(y/x)$$

Feature transform

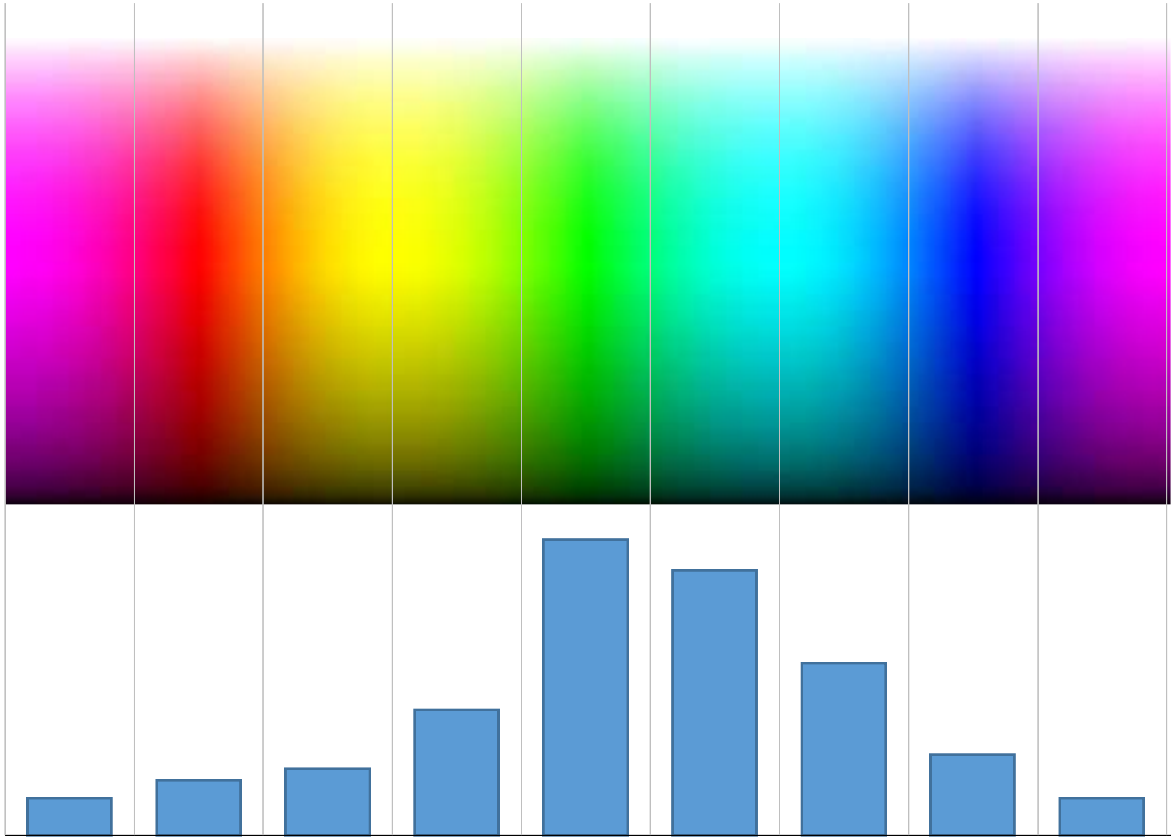
Polar space



Nonlinear classifier in original space!

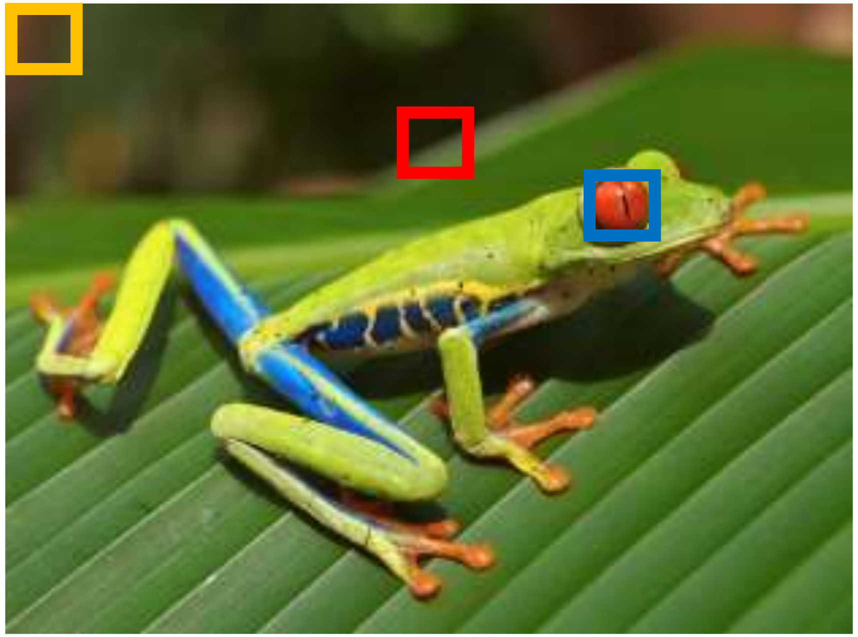
Linear classifier in feature space

Image Features: Color Histogram



Ignores texture,
spatial positions

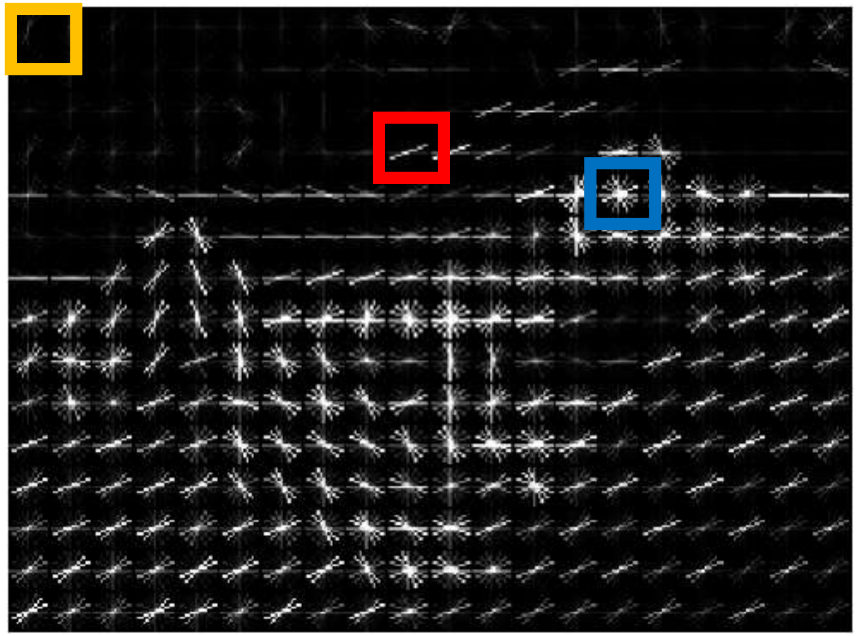
Image Features: Histogram of Oriented Gradients (HoG)



Weak edges

Strong diagonal edges

Edges in all directions



1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Captures texture and position, robust to small image changes

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

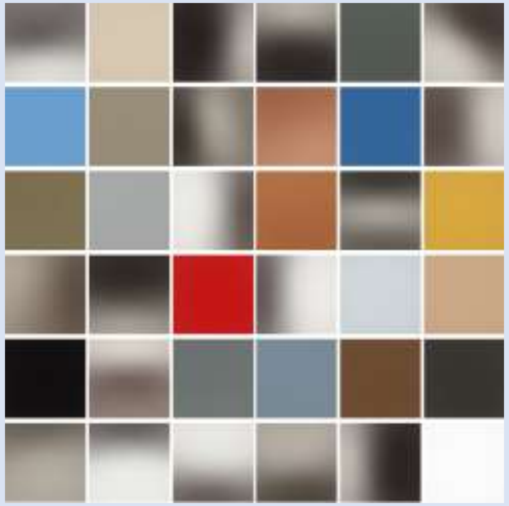
Image Features: Bag of Words (Data-Driven)

Step 1: Build codebook

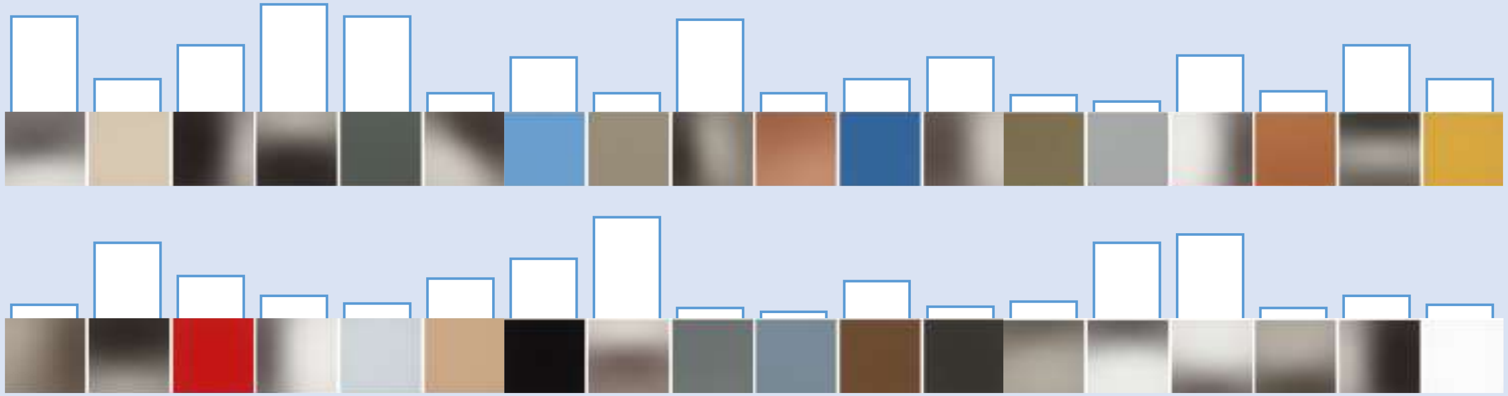
Extract random patches

from all images

Cluster patches to form "codebook" of "visual words"



Step 2: Encode images



Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005

Image Features

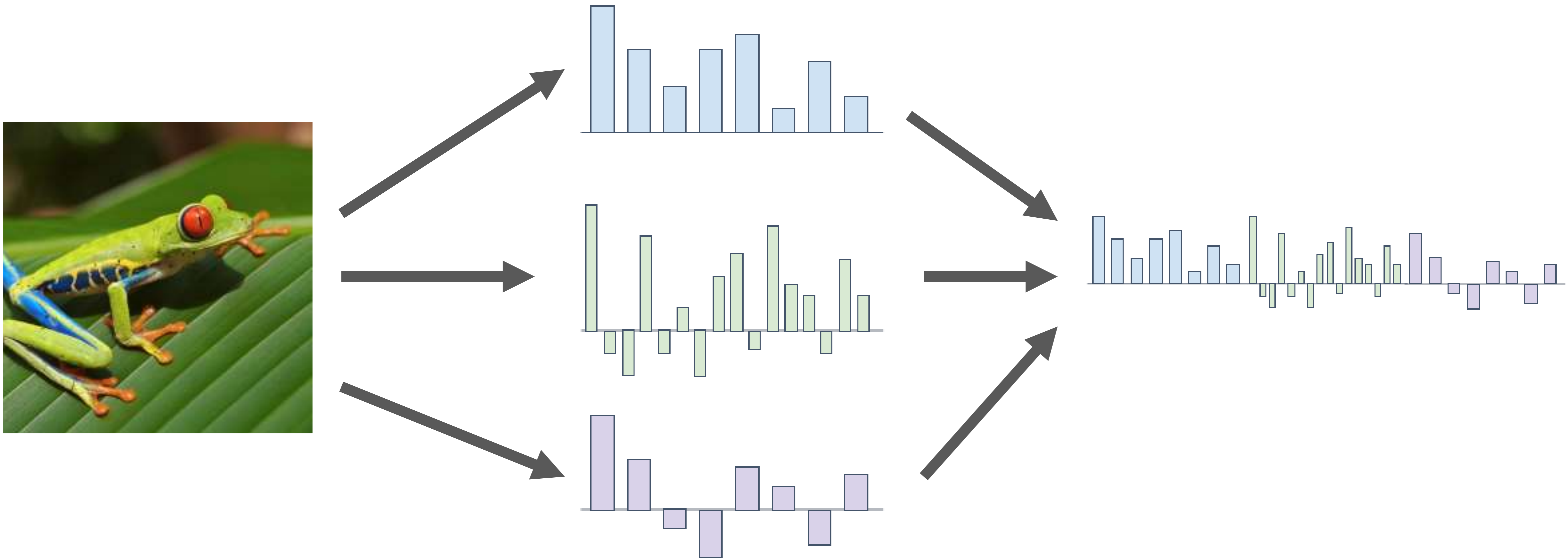
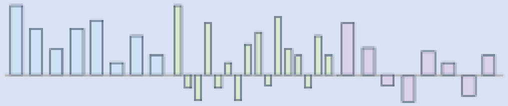


Image Features

Feature Extraction



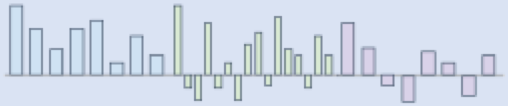
f

10 numbers giving scores for classes

training

Image Features vs Neural Networks

Feature Extraction

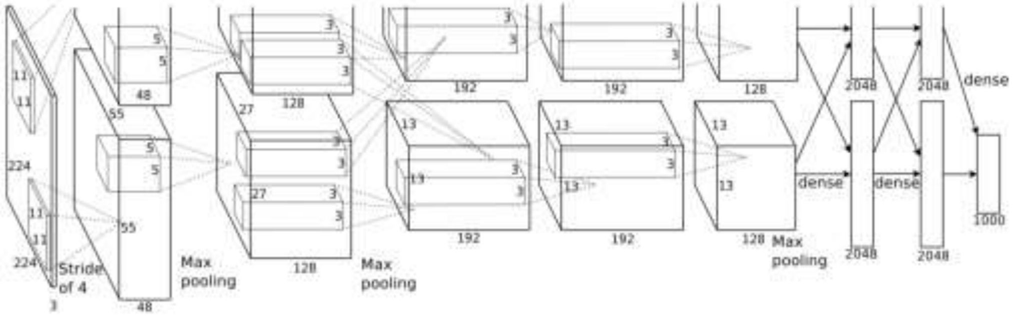


f

10 numbers giving scores for classes

training

Krizhevsky, Sutskever, and Hinton, “Imagenet classification with deep convolutional neural networks”, NIPS 2012.
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012.



10 numbers giving scores for classes

training

Neural Networks

(**Before**) Linear score function: $\boldsymbol{s = Wx}$

(**Now**) 2-layer Neural Network: $\boldsymbol{s = W_2 \max(0, W_1 x)}$

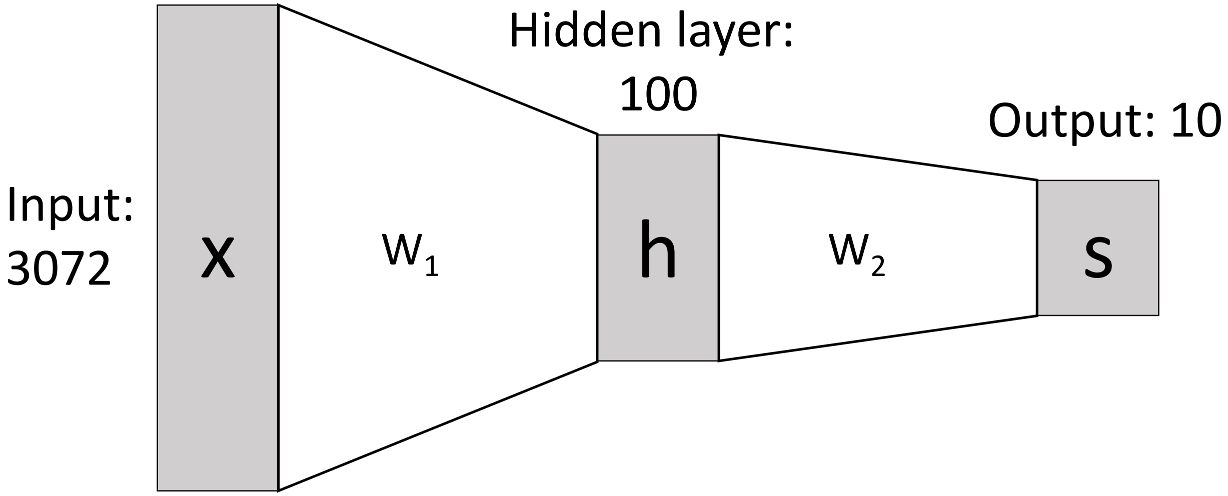
or 3-layer Neural Network:

$$\boldsymbol{s = W_3 \max(0, W_2 \max(0, W_1 x))}$$

Neural Networks

(**Before**) Linear score function: $\boldsymbol{s = Wx}$

(**Now**) 2-layer Neural Network: $\boldsymbol{s = W_2 \max(0, W_1 x)}$

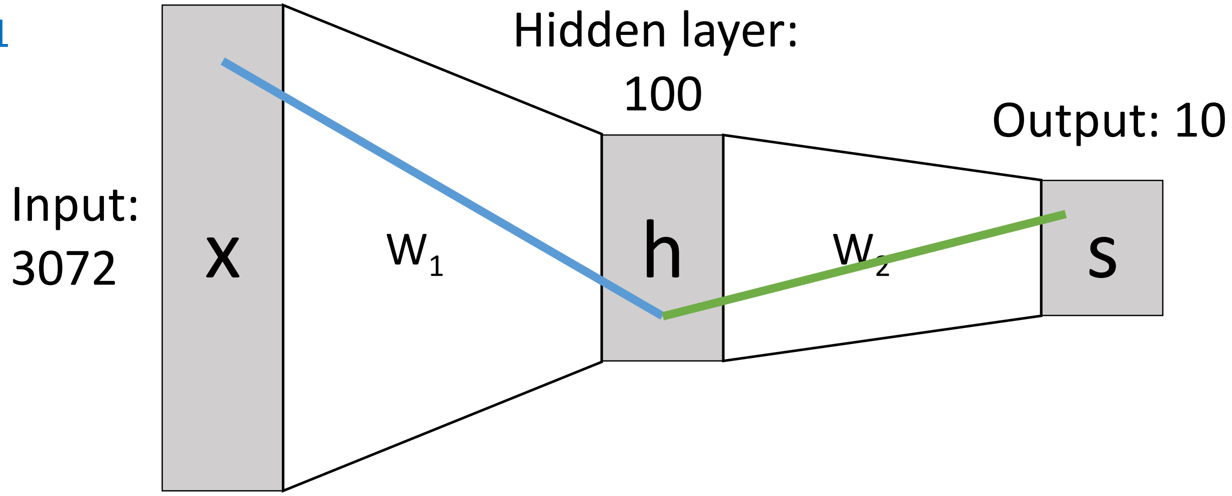


Neural Networks

(**Before**) Linear score function: $\boldsymbol{s = Wx}$

(**Now**) 2-layer Neural Network: $\boldsymbol{s = W_2 \max(0, W_1 x)}$

Element (i, j) of W_1 gives the effect on h_i from x_j



Element (i, j) of W_2 gives the effect on s_i from h_j

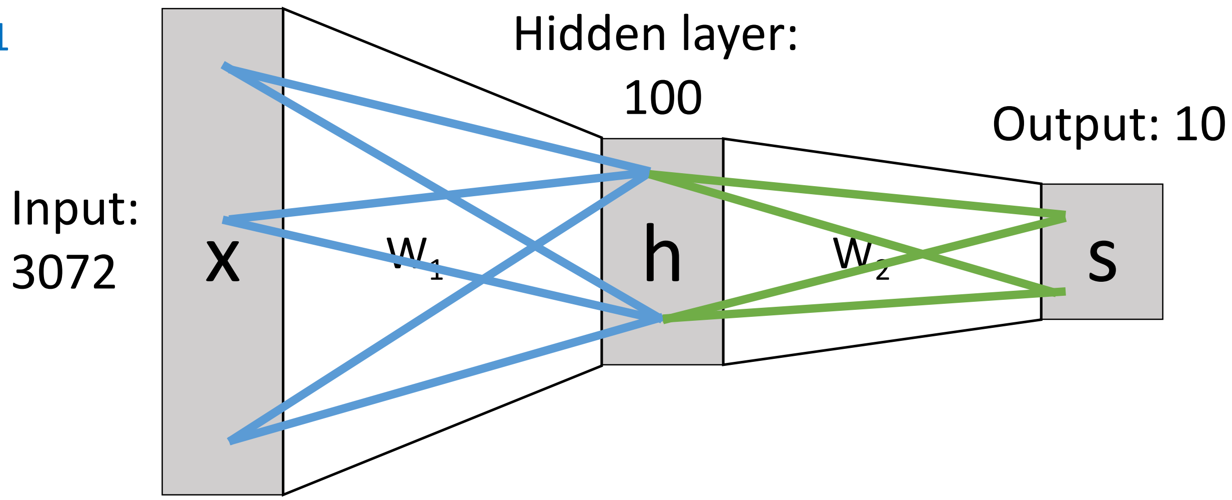
Neural Networks

(**Before**) Linear score function: $\boldsymbol{s = Wx}$

(**Now**) 2-layer Neural Network: $\boldsymbol{s = W_2 \max(0, W_1 x)}$

Element (i, j) of W_1 gives the effect on h_i from x_j

All elements of x affect all elements of h



Element (i, j) of W_2 gives the effect on s_i from h_j

All elements of h affect all elements of s

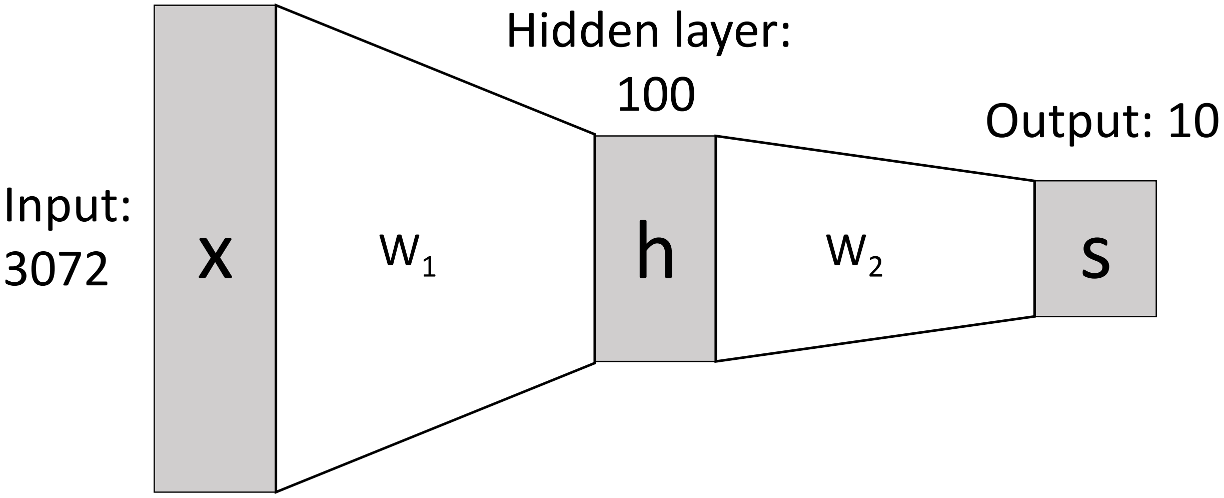
Fully-connected neural network
Also “Multi-Layer Perceptron” (MLP)

Neural Networks

Linear classifier: One template per class

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network:

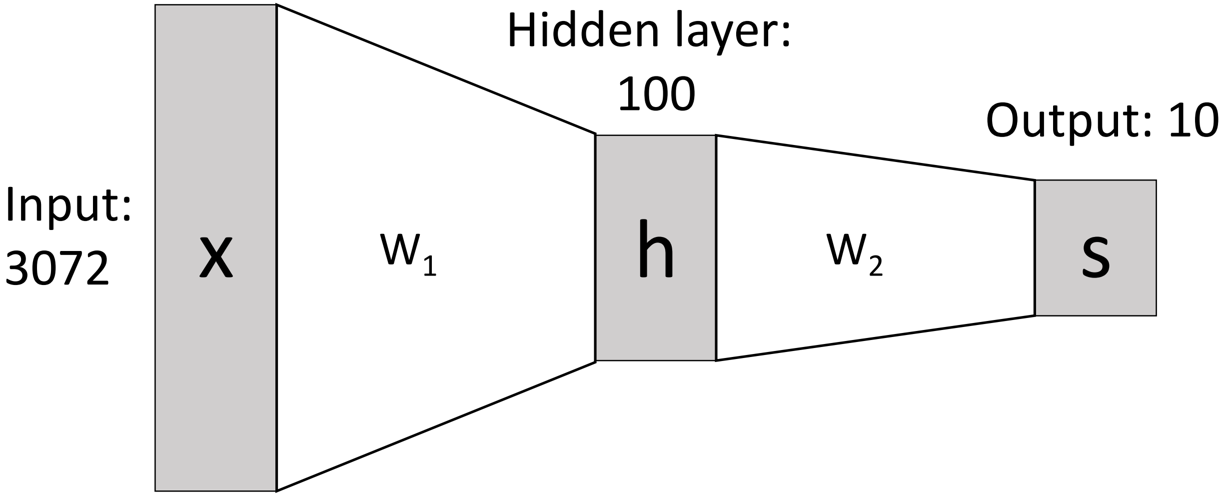


Neural Networks

Neural net: first layer is bank of templates;
Second layer recombines templates

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network:

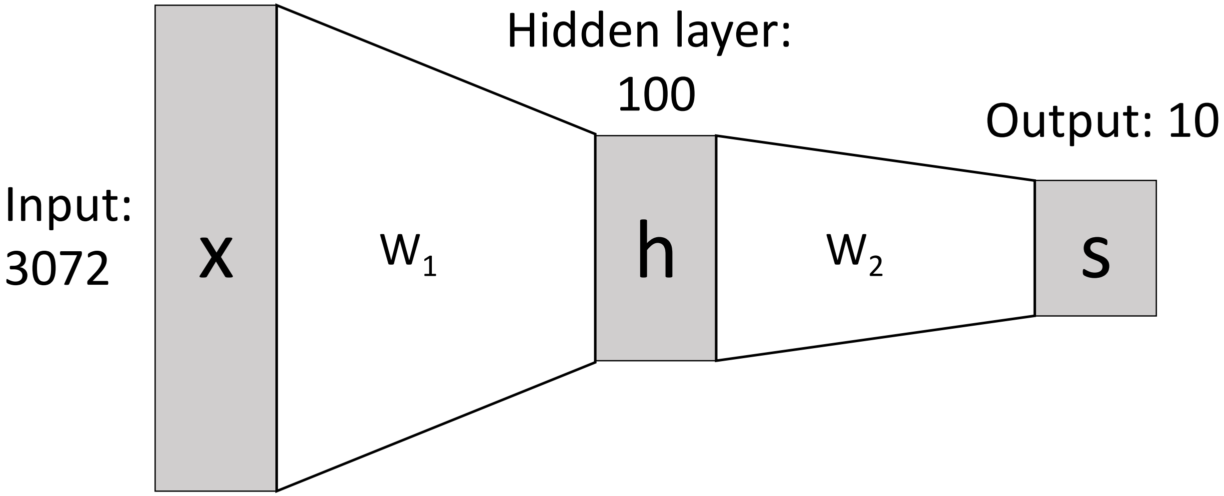


Neural Networks

Can use different templates to cover multiple modes of a class!

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network:

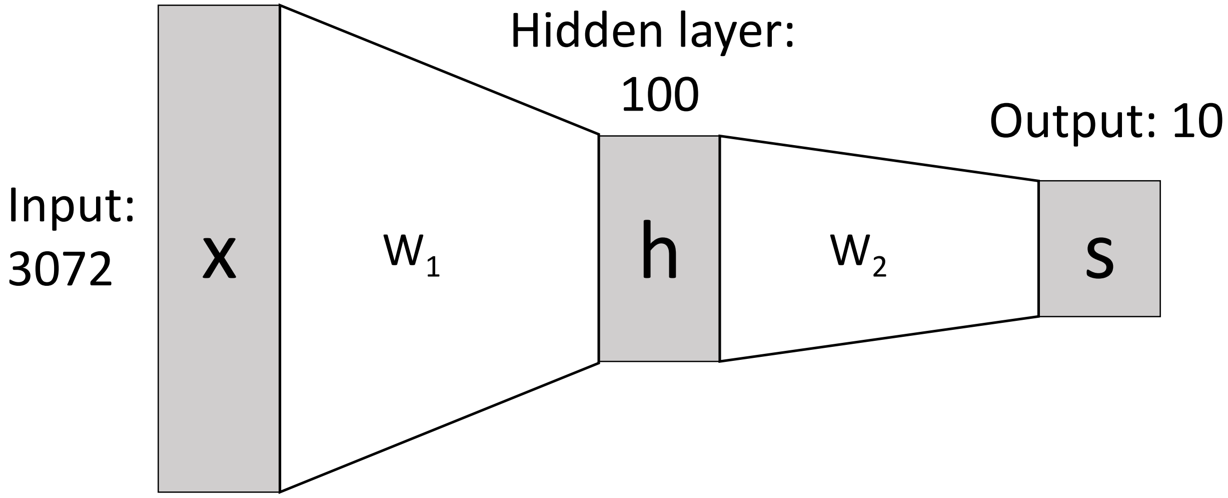


Neural Networks

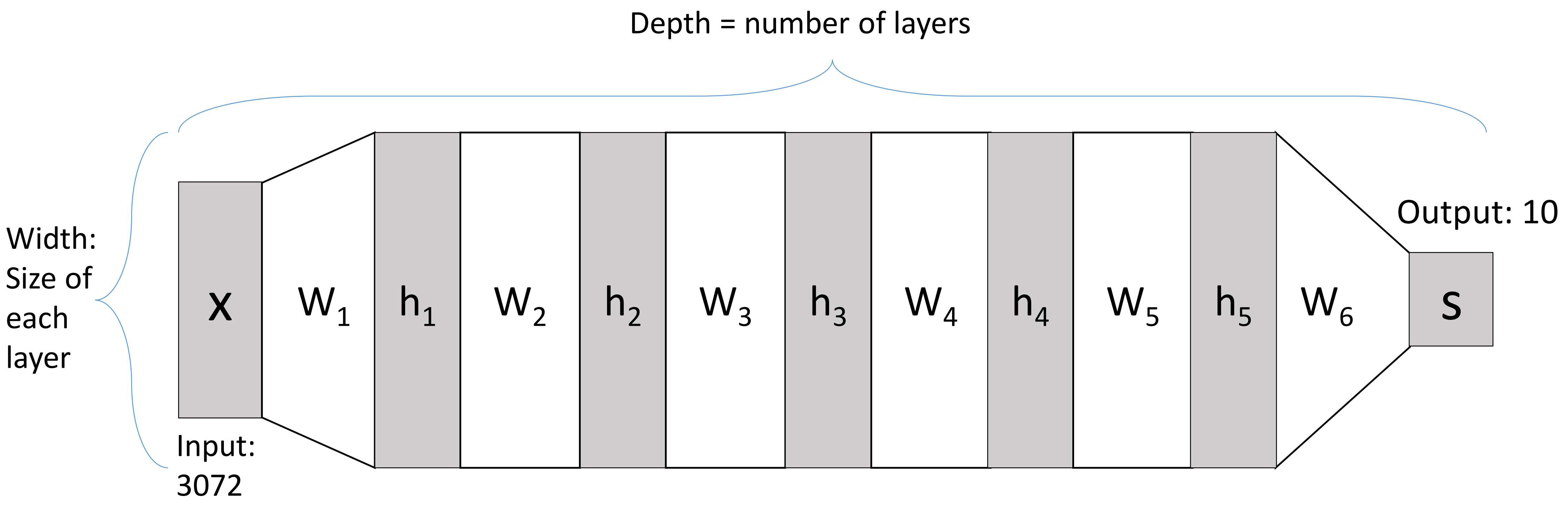
"Distributed representation":
Most templates not interpretable!

(Before) Linear score function:

(Now) 2-layer Neural Network:



Deep Neural Networks



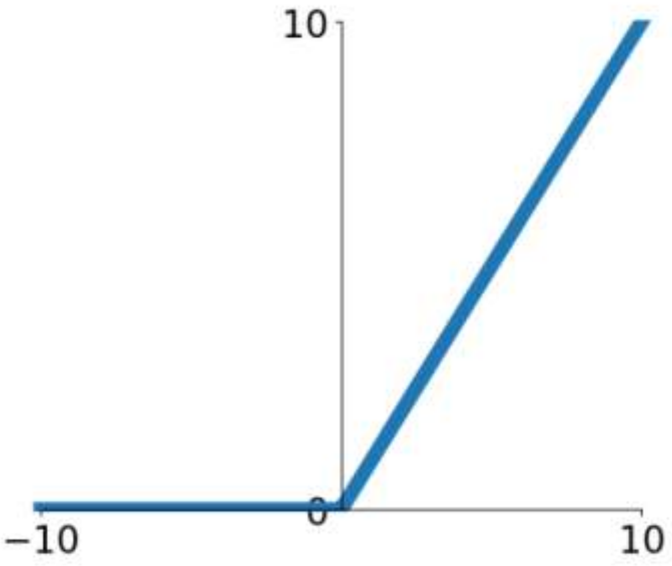
$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Activation Functions

2-layer Neural Network

$$s = W_2 \max(0, W_1 x)$$

The function $\mathrm{ReLU}(z) = \max(0, z)$ is called “Rectified Linear Unit”



This is called the **activation function** of the neural network

Q: What happens if we build a neural network with no activation function?

$$s = W_2\, W_1\, x$$

$$W_3 = W_2\, W_1 \qquad s = W_3\, x$$

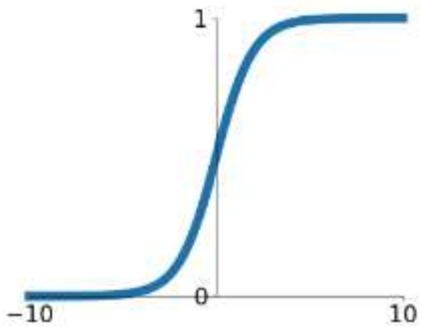
A: We end up with a linear classifier!

Activation Functions

ReLU is a good default choice for most problems

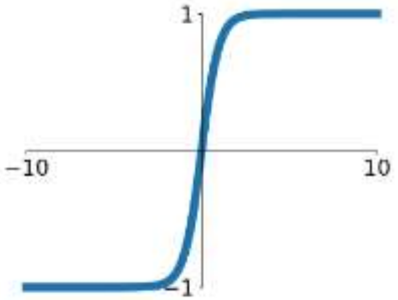
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



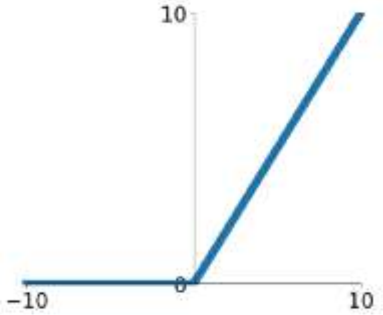
tanh

$\tanh(x)$



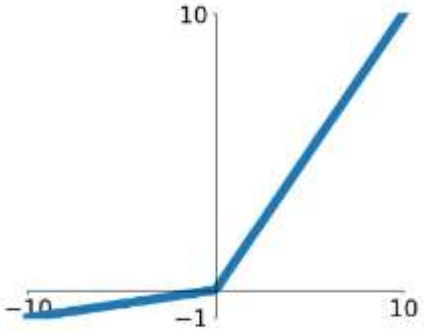
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

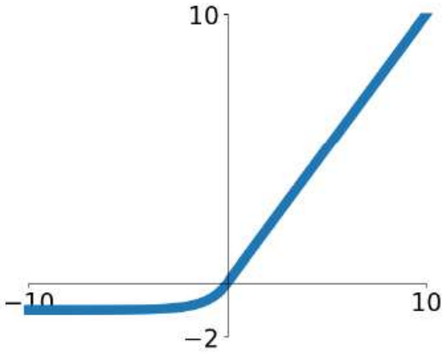


Maxout

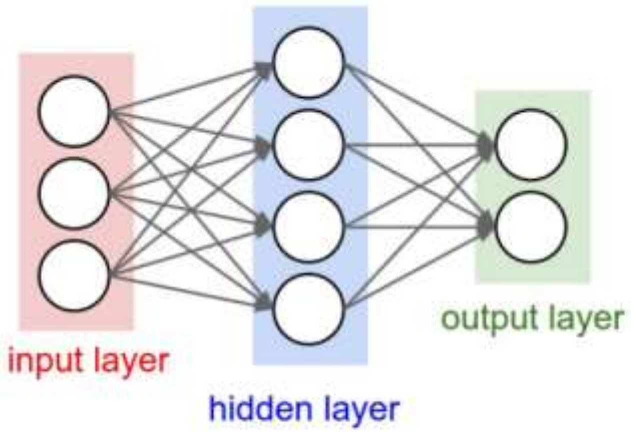
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Net in <20 lines!




```python
from tensorflow import keras
from tensorflow.keras import layers

num_classes = 10
input_shape = (28, 28, 1)
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.1)
```

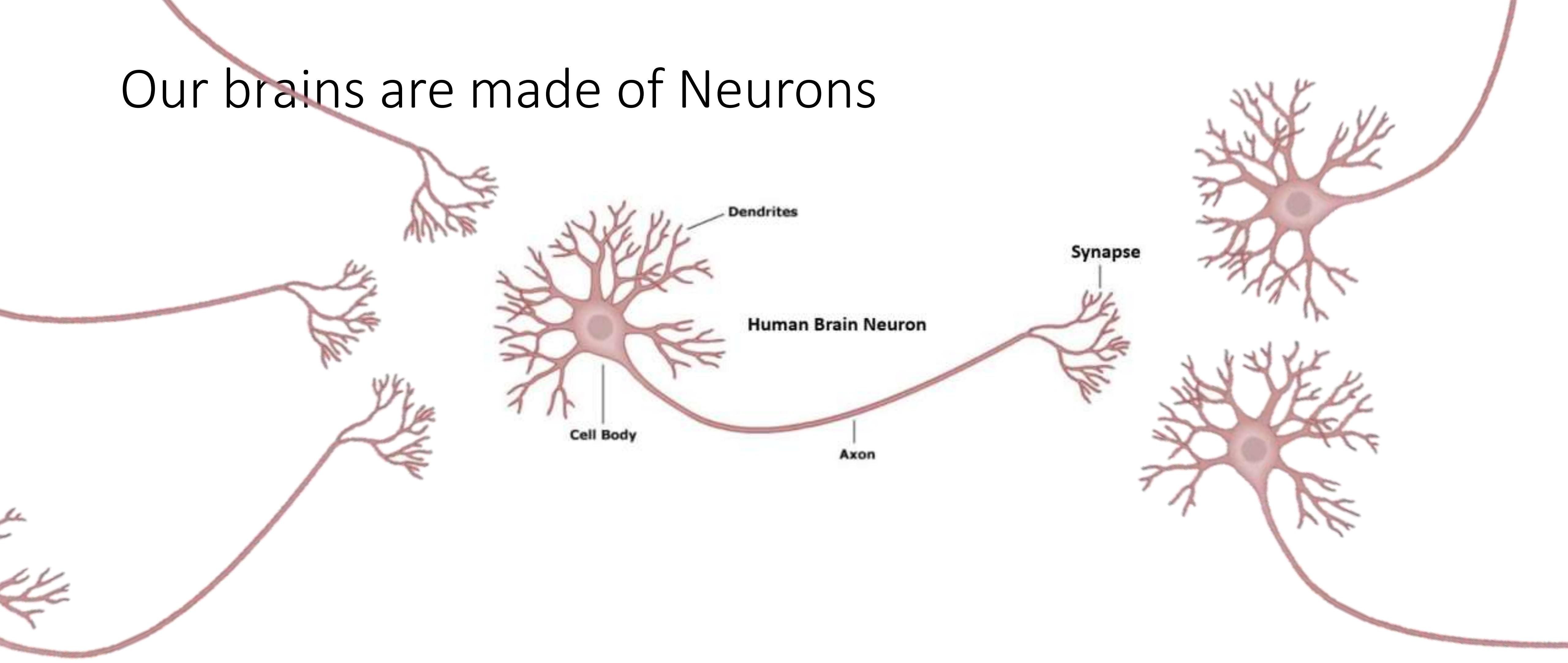
Initialize network and load data

Define layers size and depth.
Use relu and softmax activation.

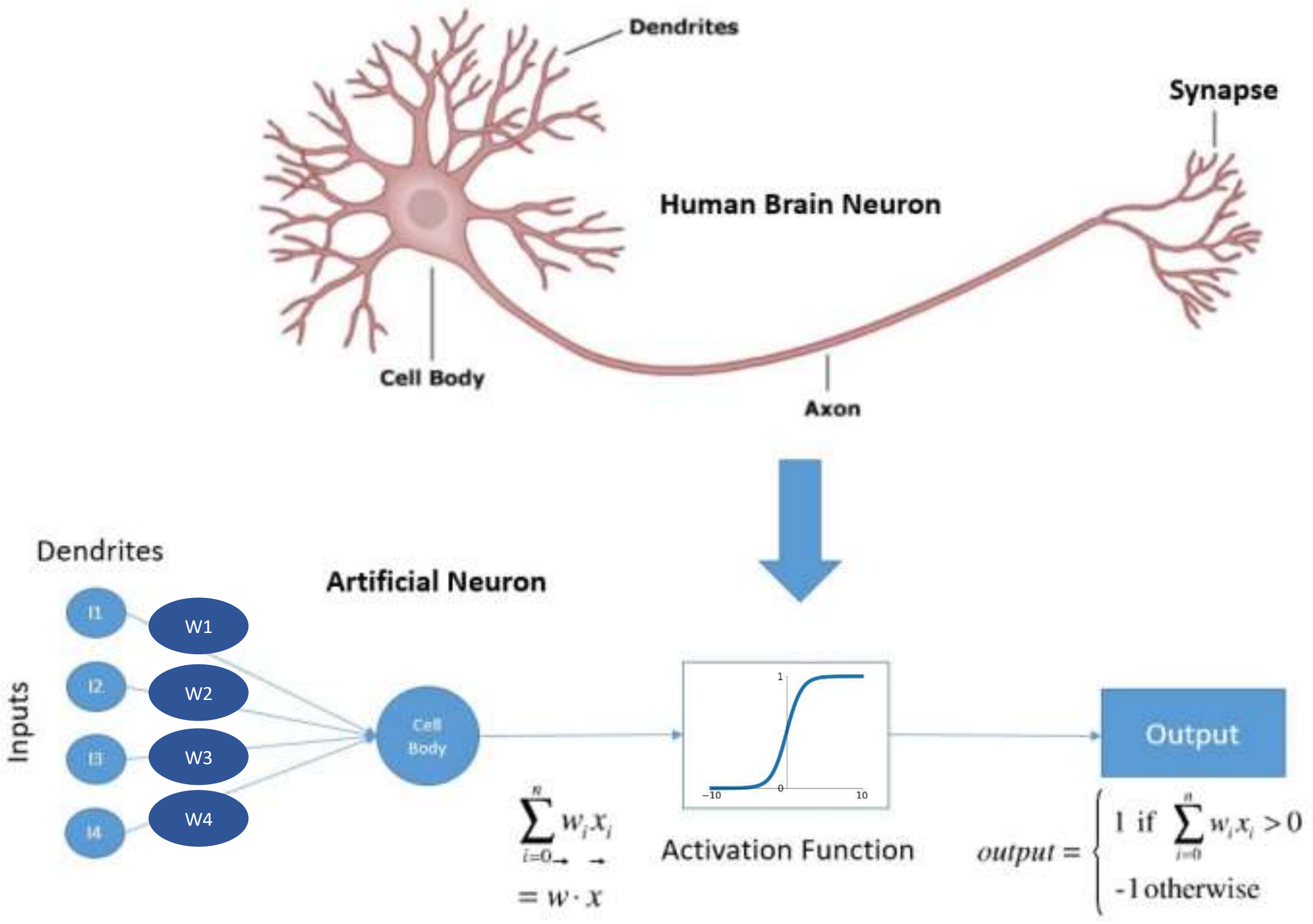
Select loss and gradient optimizer methods

Select batch size, epochs and data split, then train (fit)

Our brains are made of Neurons



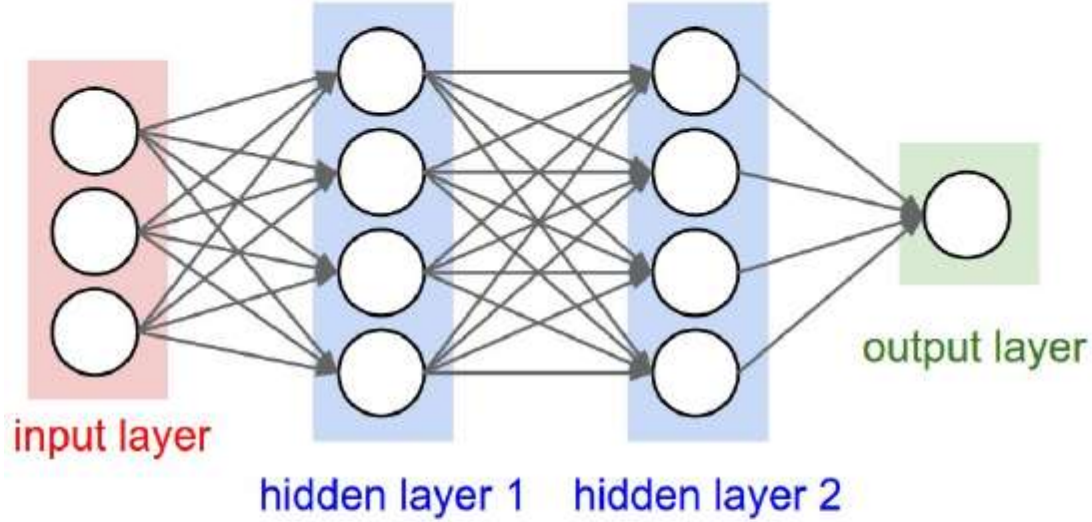
Our brains are made of Neurons



Biological Neurons:
Complex connectivity patterns

https://www.scientificamerican.com/article/scientists-surprised-to-find-no-two-neurons-are-genetically-alike/

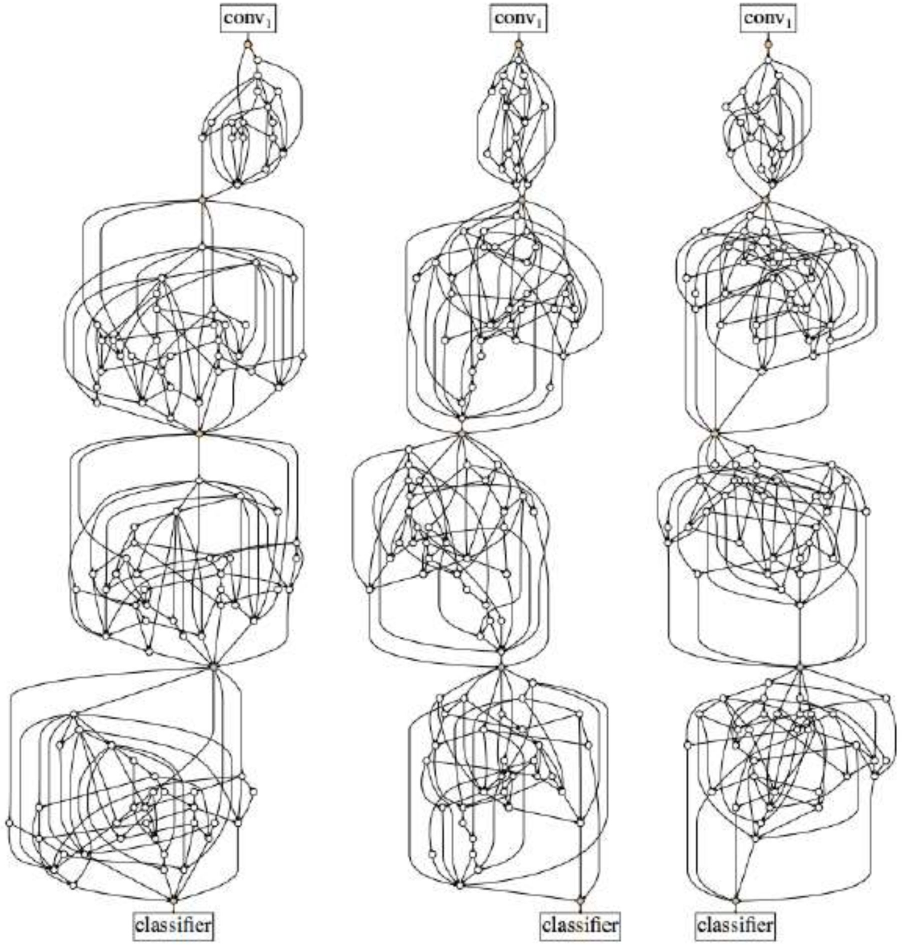
Neurons in a neural network:
Organized into regular layers for computational efficiency



Biological Neurons:
Complex connectivity patterns

https://www.scientificamerican.com/article/scientists-surprised-to-find-no-two-neurons-are-genetically-alike/

But neural networks with random connections can work too!



Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”, ICCV 2019

Be very careful with brain analogies!

Biological Neurons:
Complex connectivity patterns

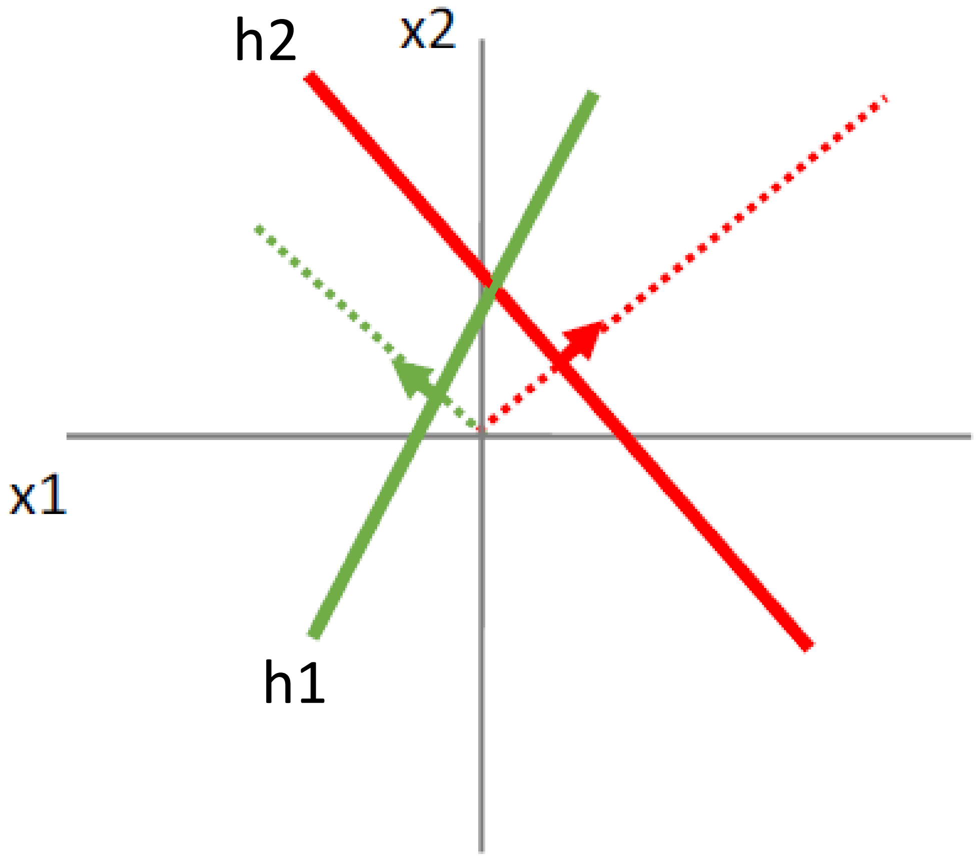
https://www.scientificamerican.com/article/scientists-surprised-to-find-no-two-neurons-are-genetically-alike/

Biological Neurons:

- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex nonlinear dynamical system

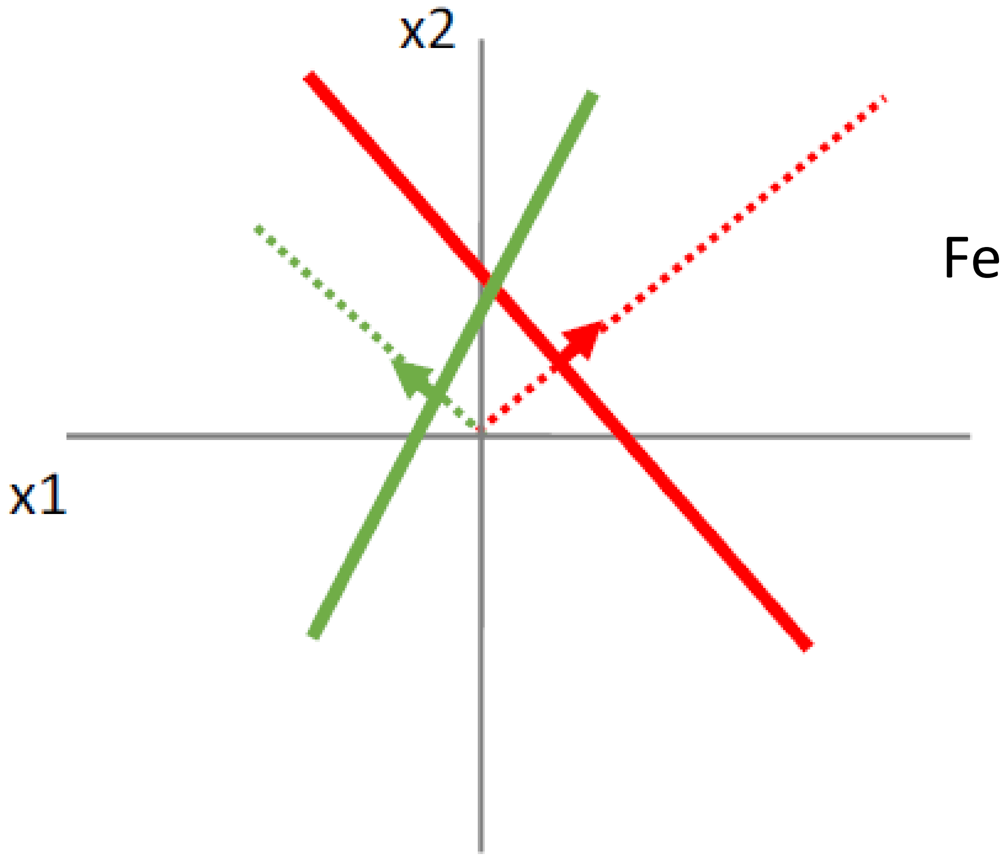
Space Warping

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

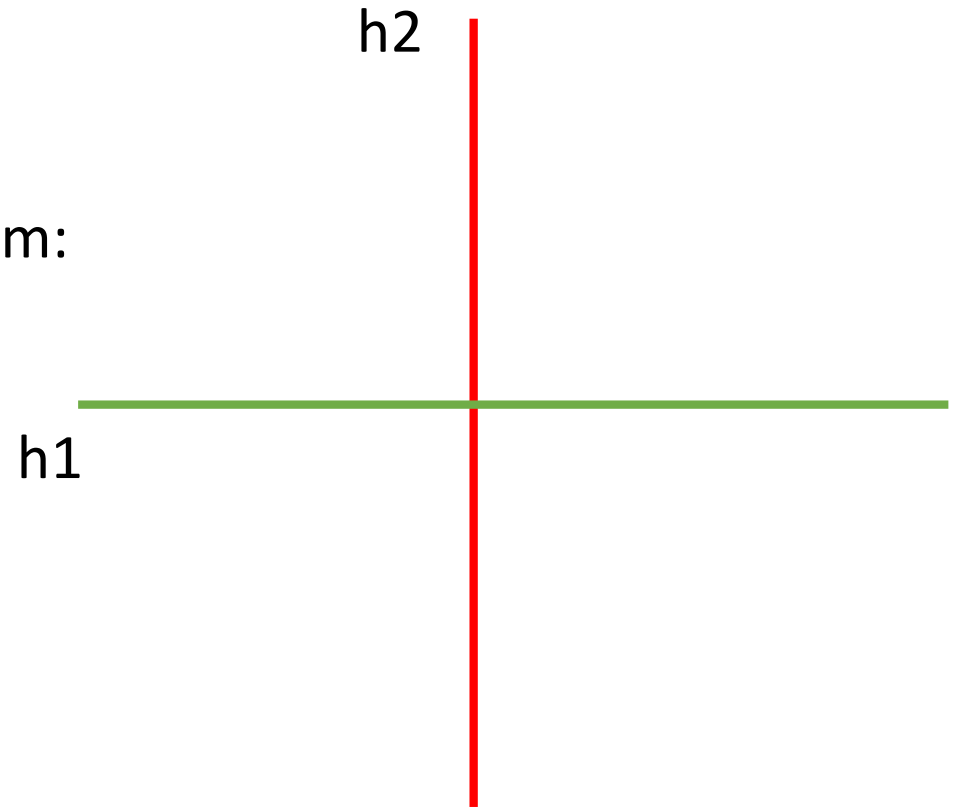


Space Warping

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

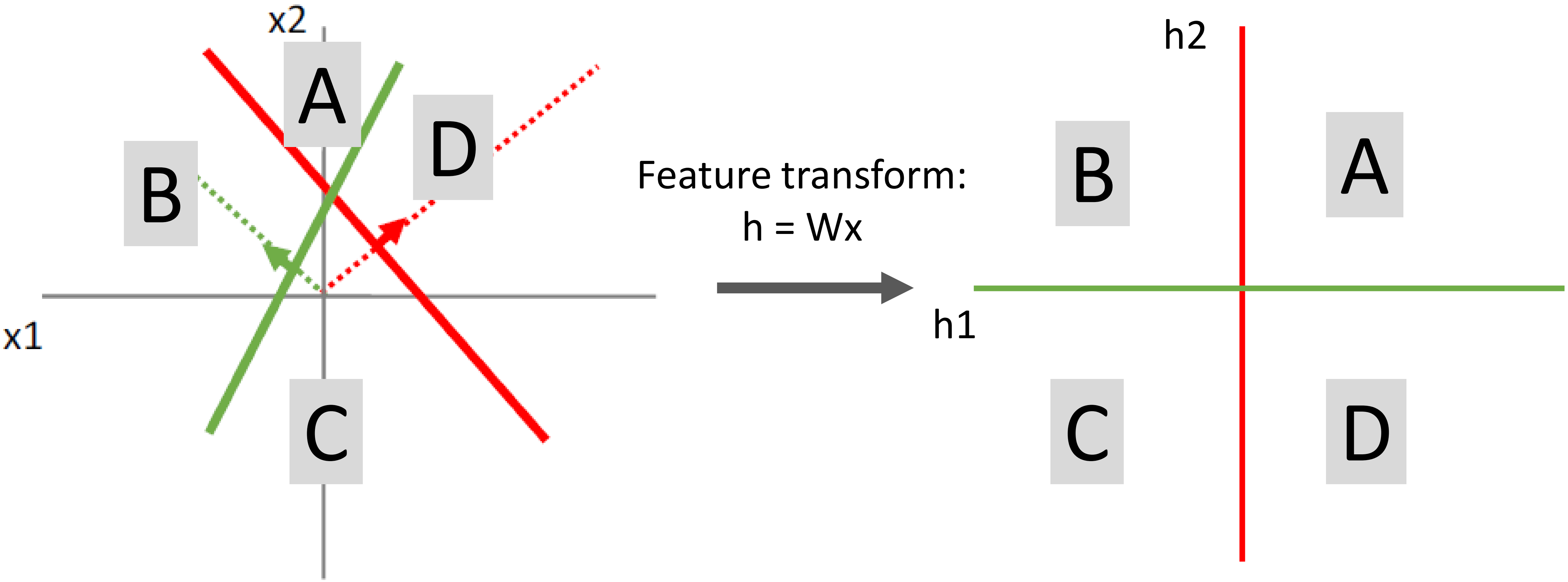


Feature transform:
h = Wx



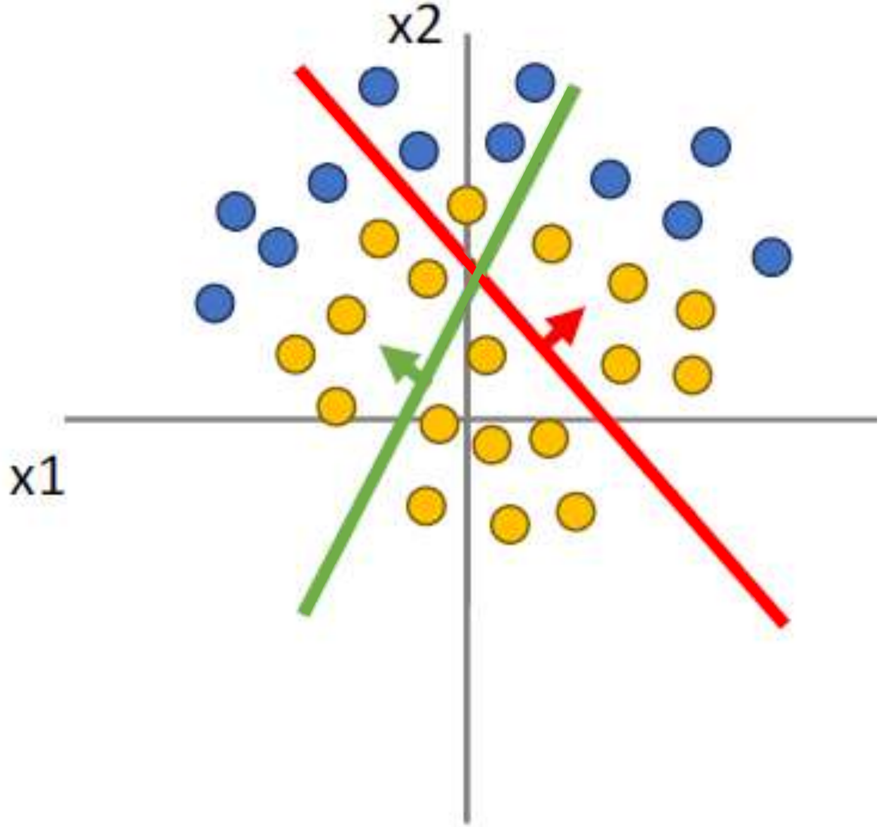
Space Warping

Consider a linear transform: h = Wx
Where x, h are both 2-dimensional



Space Warping

Points not linearly separable in original space

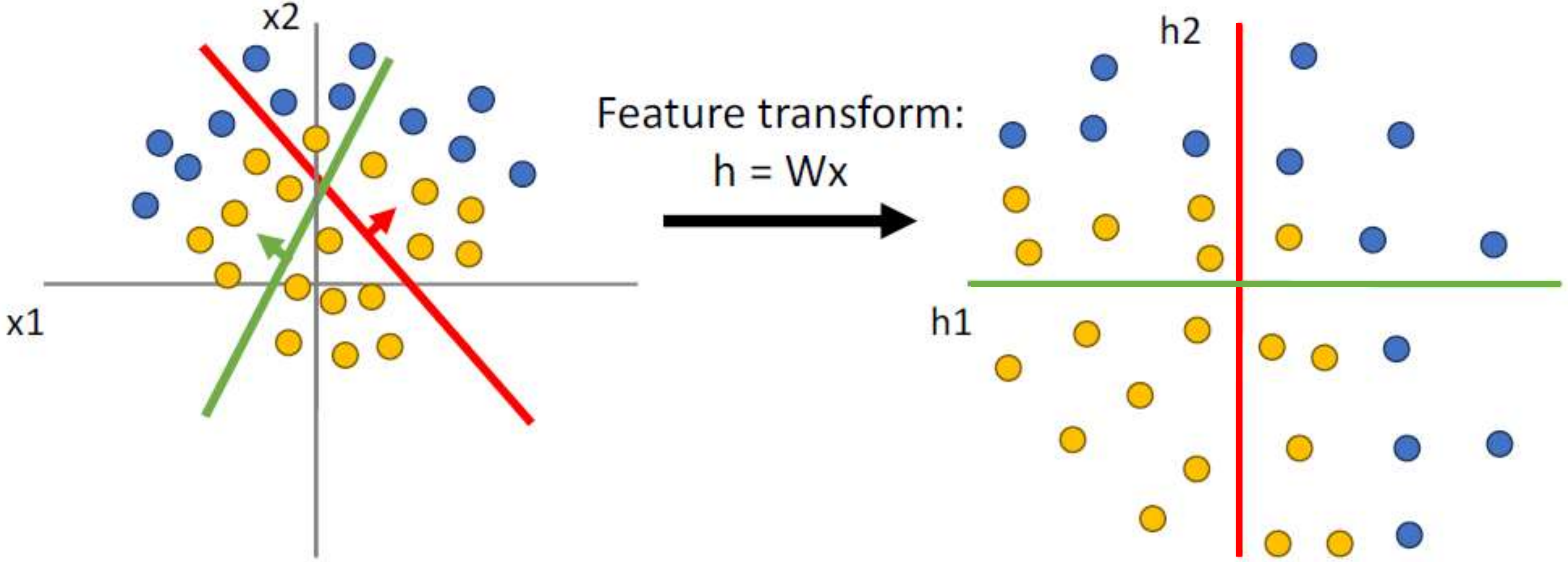


Consider a linear transform: h = Wx
Where x, h are both 2-dimensional

Space Warping

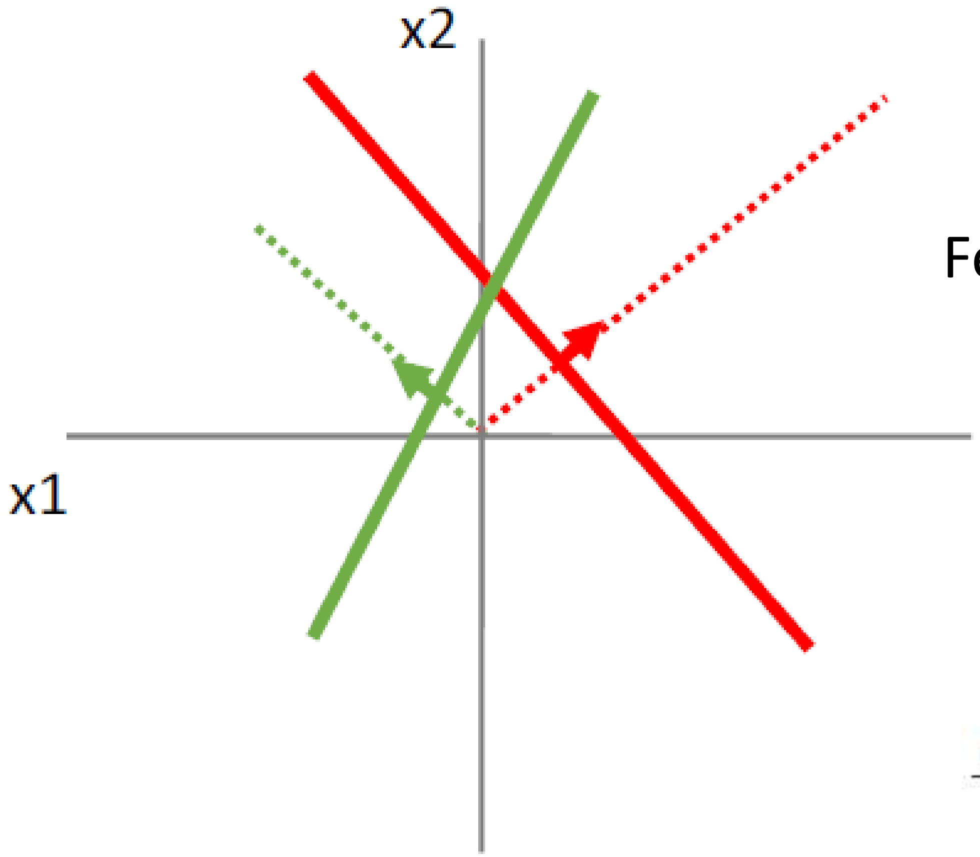
Points not linearly separable in original space

Consider a linear transform: $h = Wx$
Where x, h are both 2-dimensional

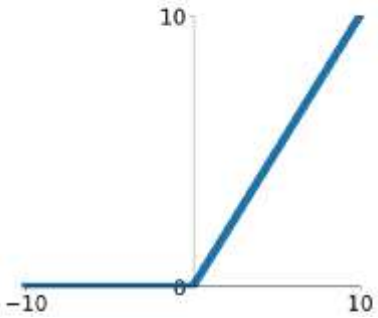


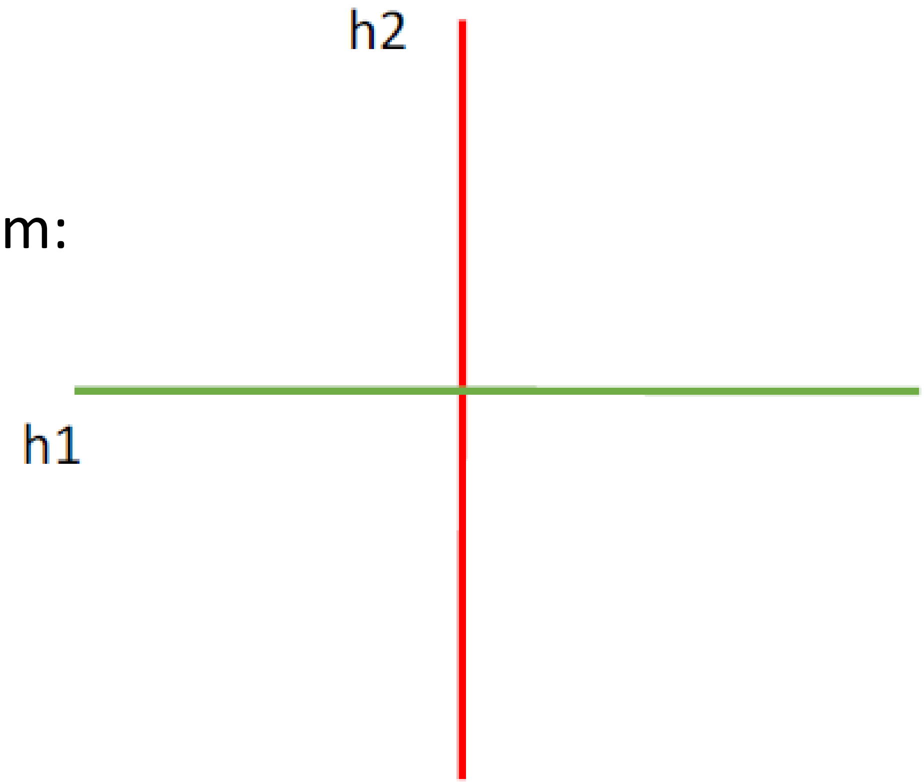
Space Warping

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



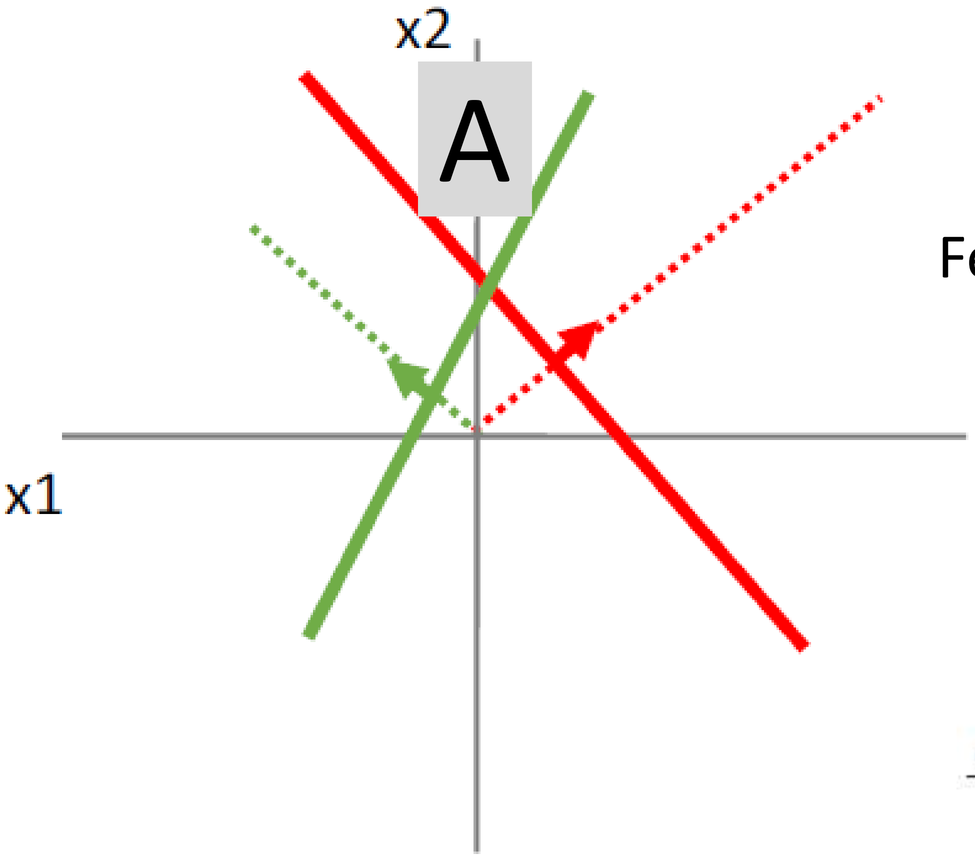
Feature transform:
h = ReLU(Wx)



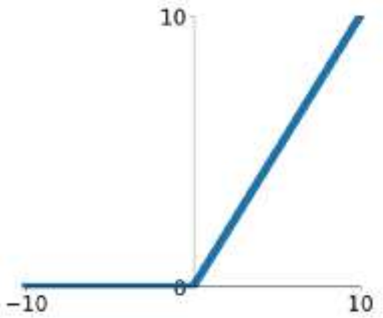


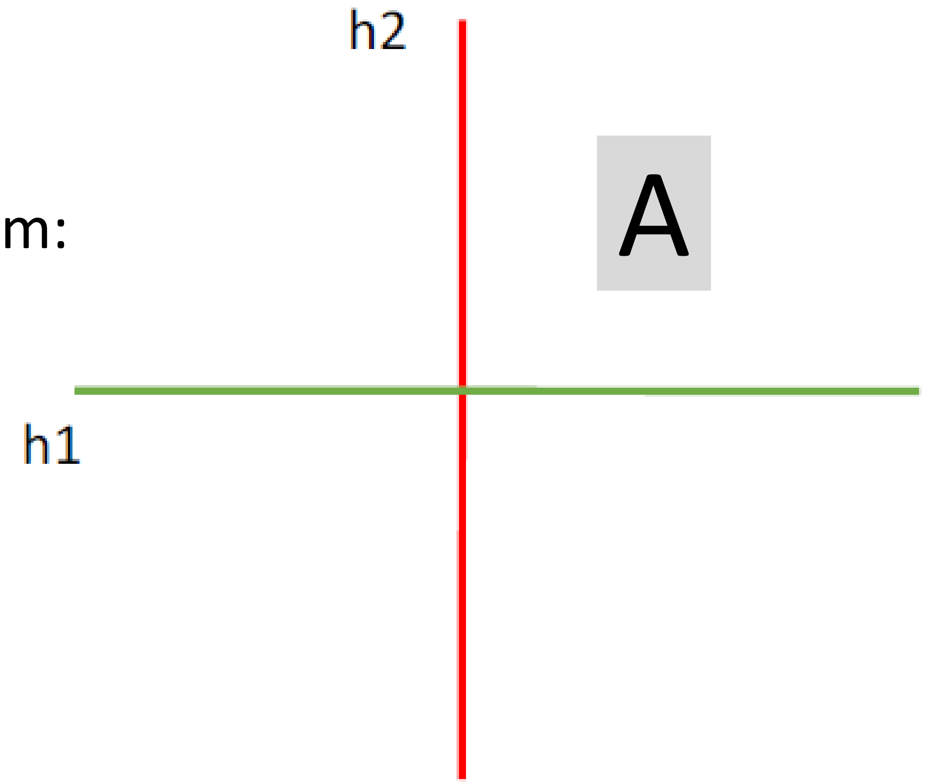
Space Warping

Consider a neural net hidden layer:
$h = \mathrm{ReLU}(Wx) = \max(0, Wx)$
Where x, h are both 2-dimensional



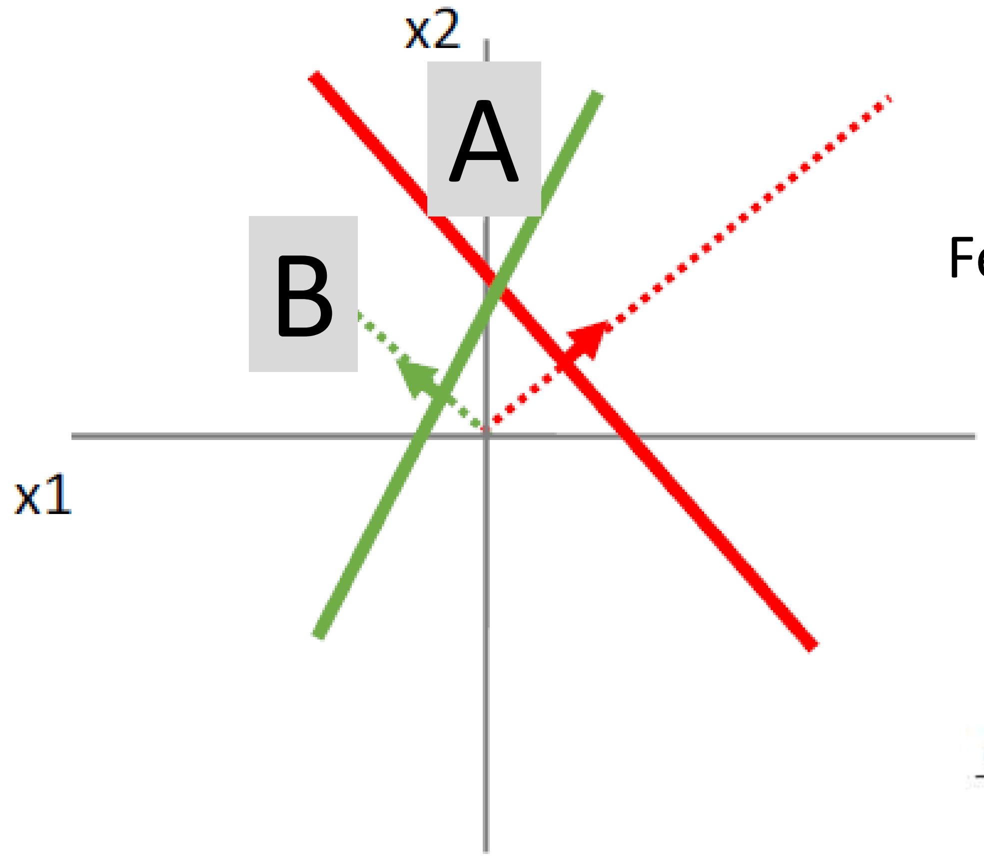
Feature transform:
$h = \mathrm{ReLU}(Wx)$



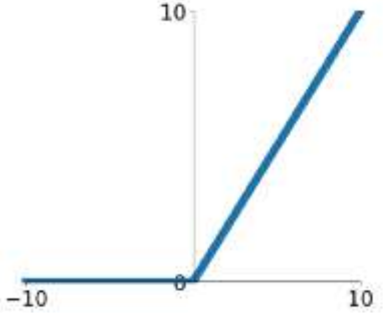


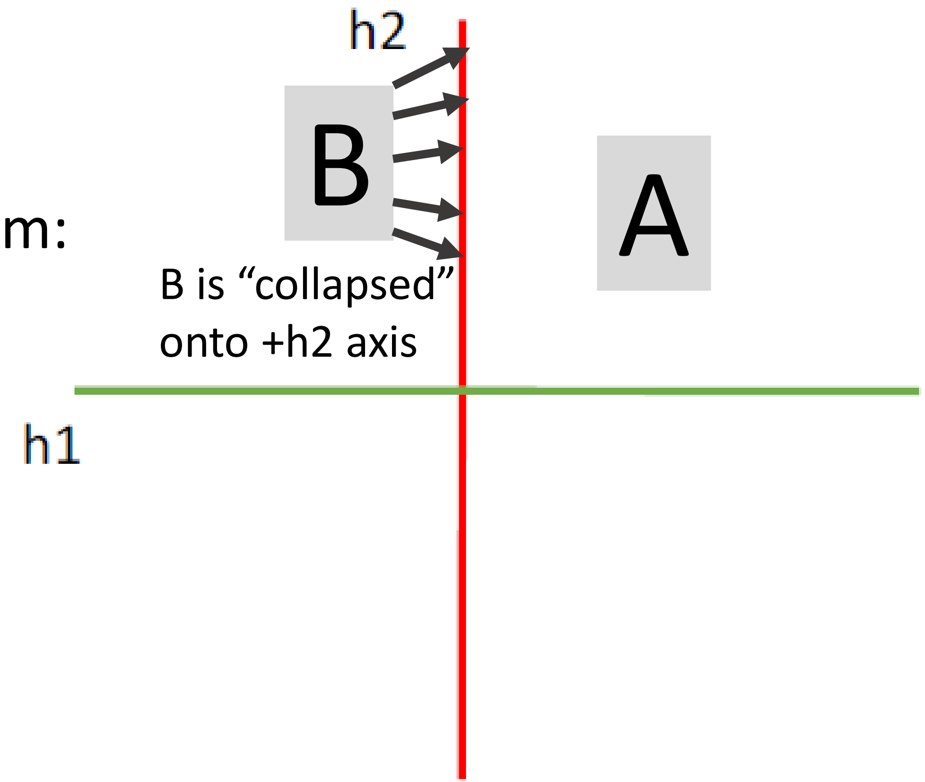
Space Warping

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



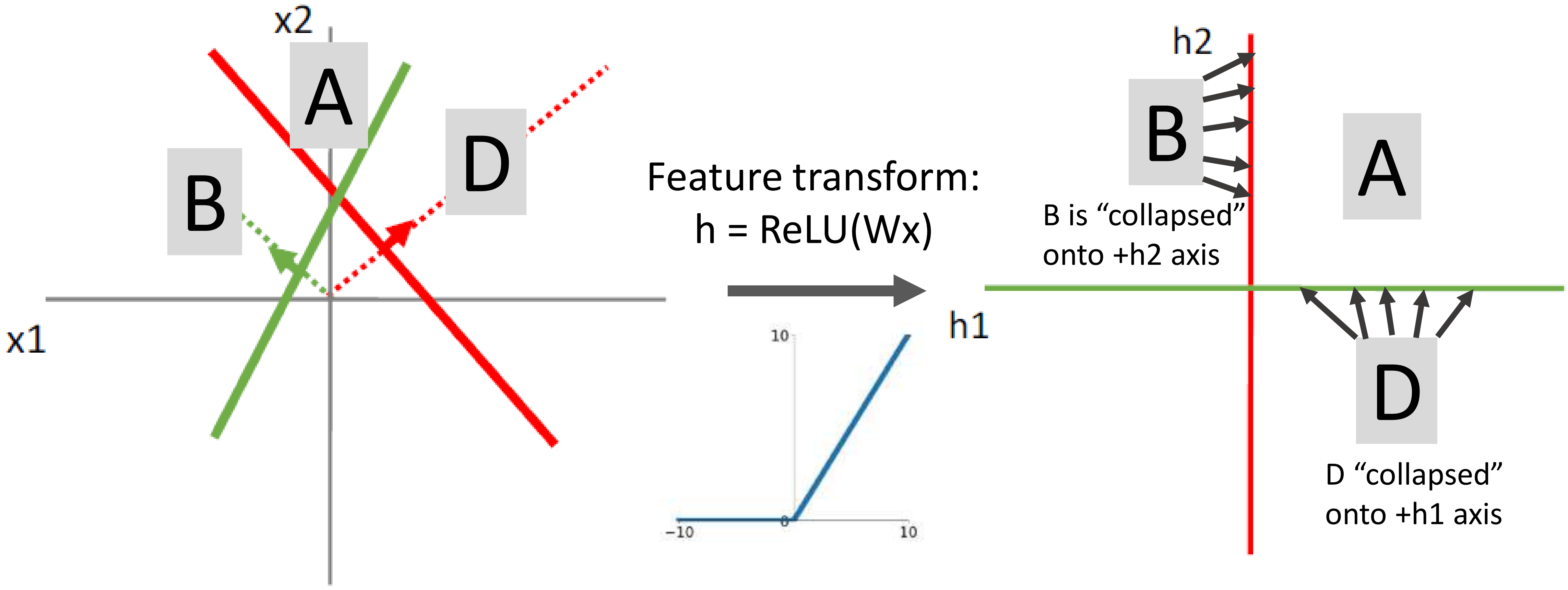
Feature transform:
h = ReLU(Wx)





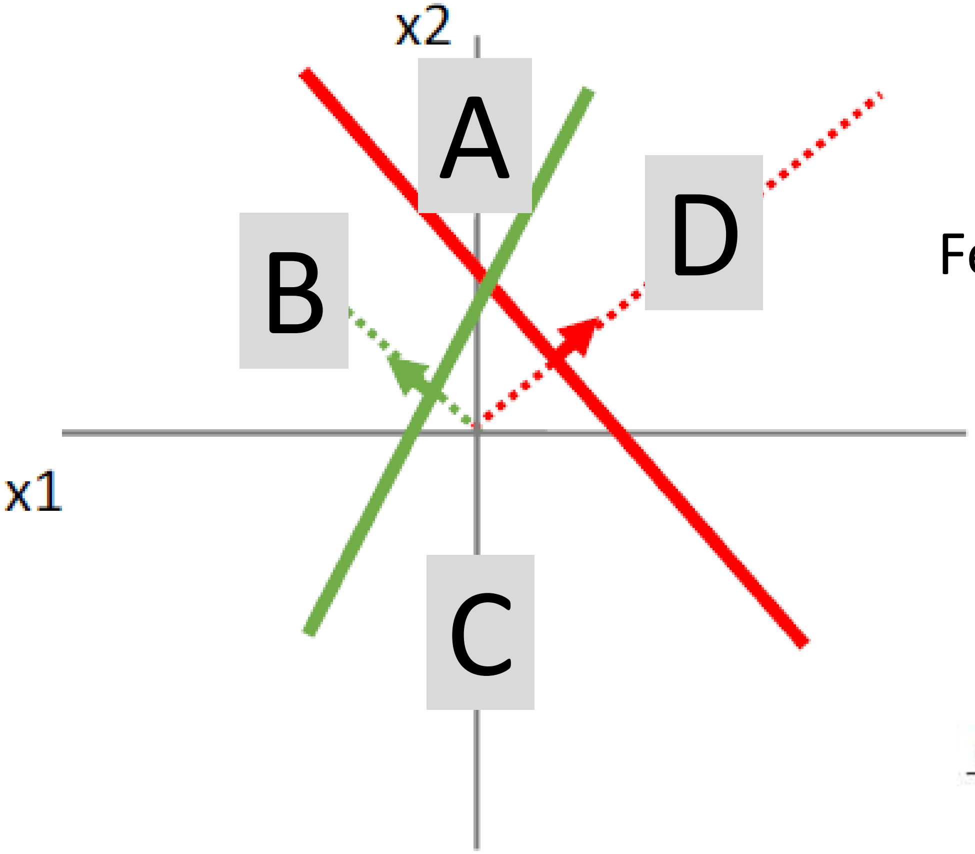
Space Warping

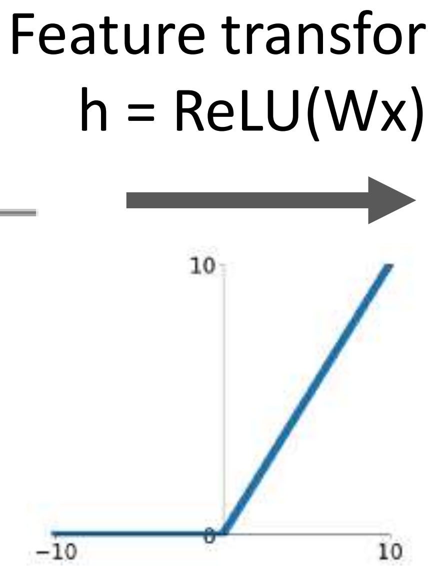
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional

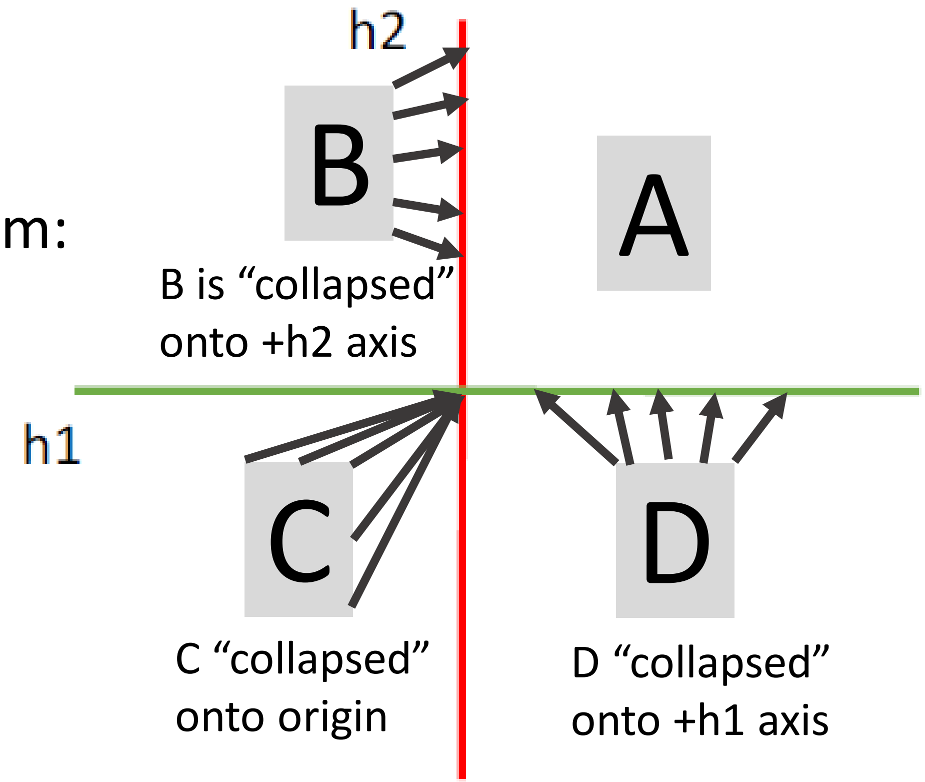


Space Warping

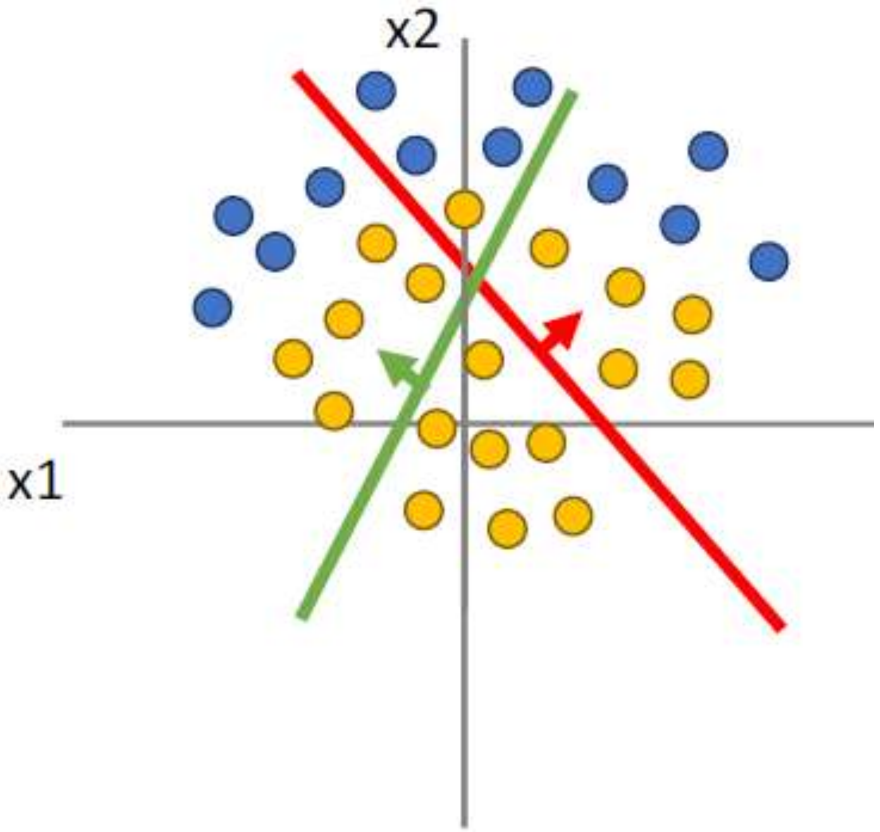
Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



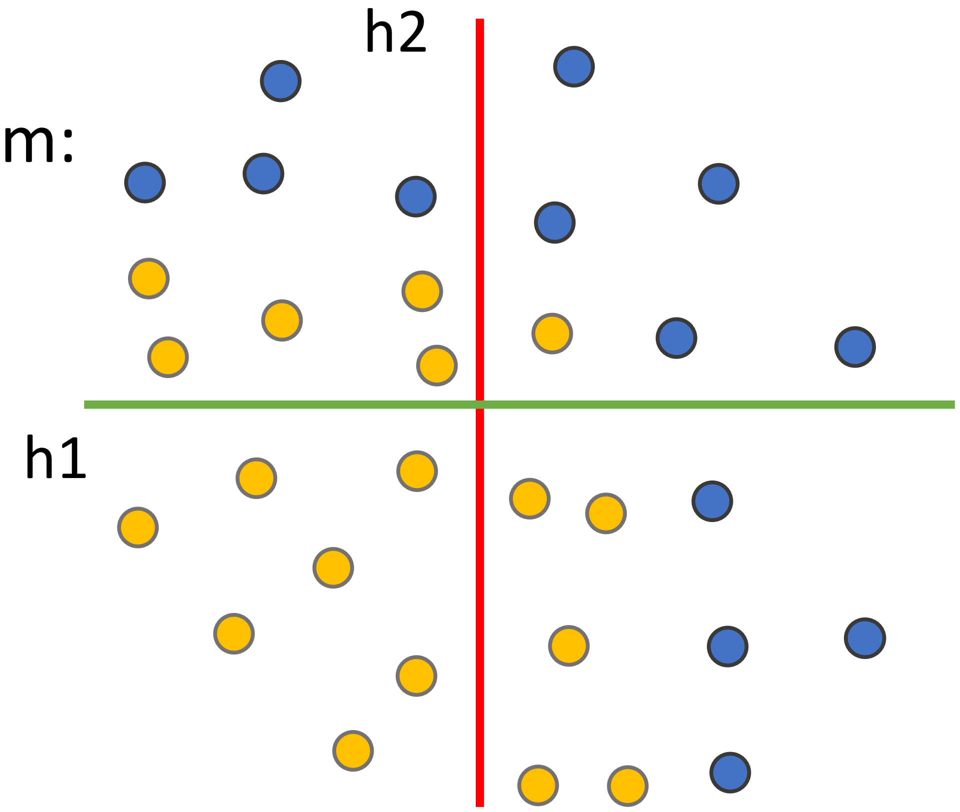




Space Warping

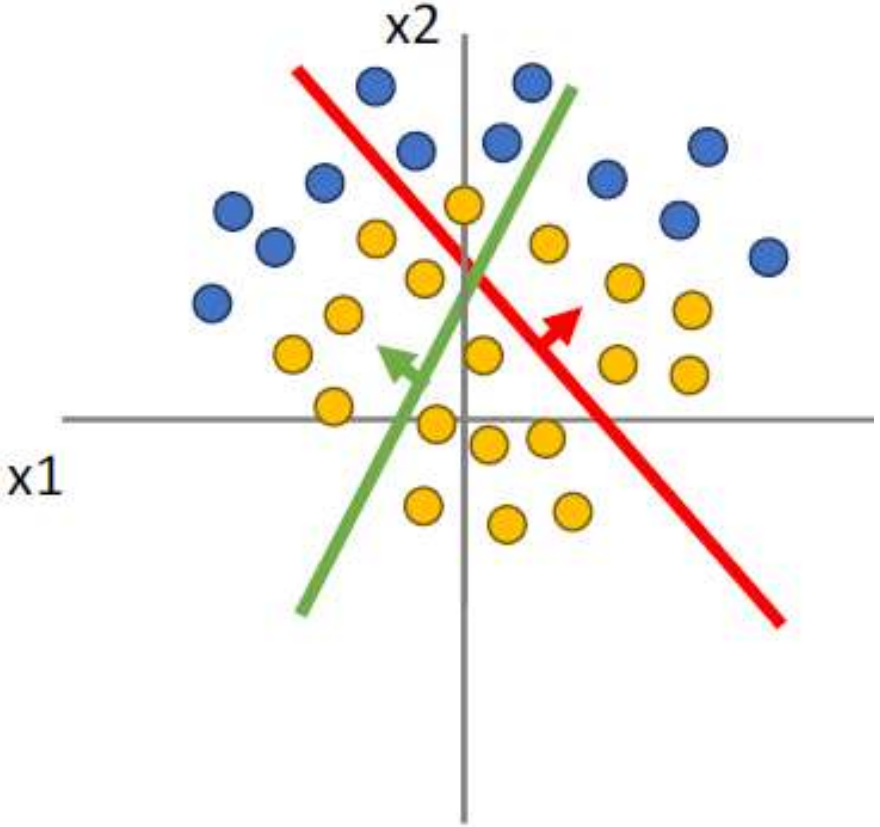


Feature transform:
$h = Wx$

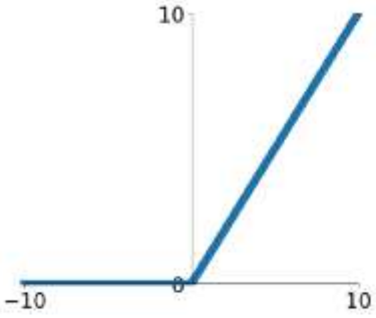


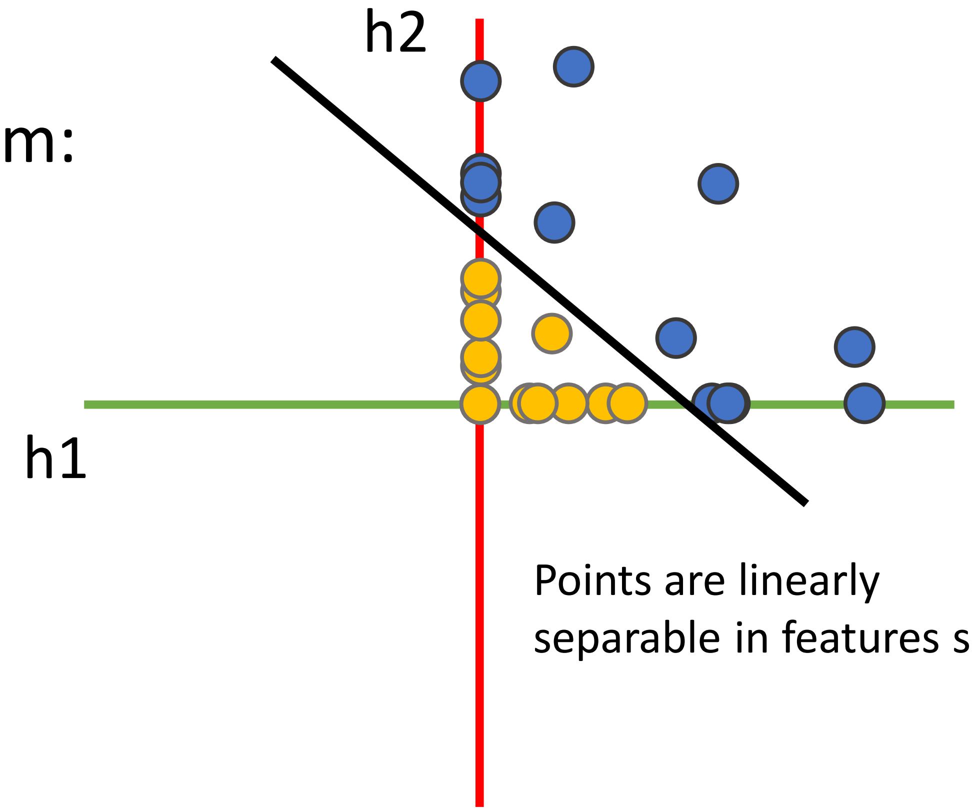
Space Warping

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



Feature transform:
h = ReLU(Wx)

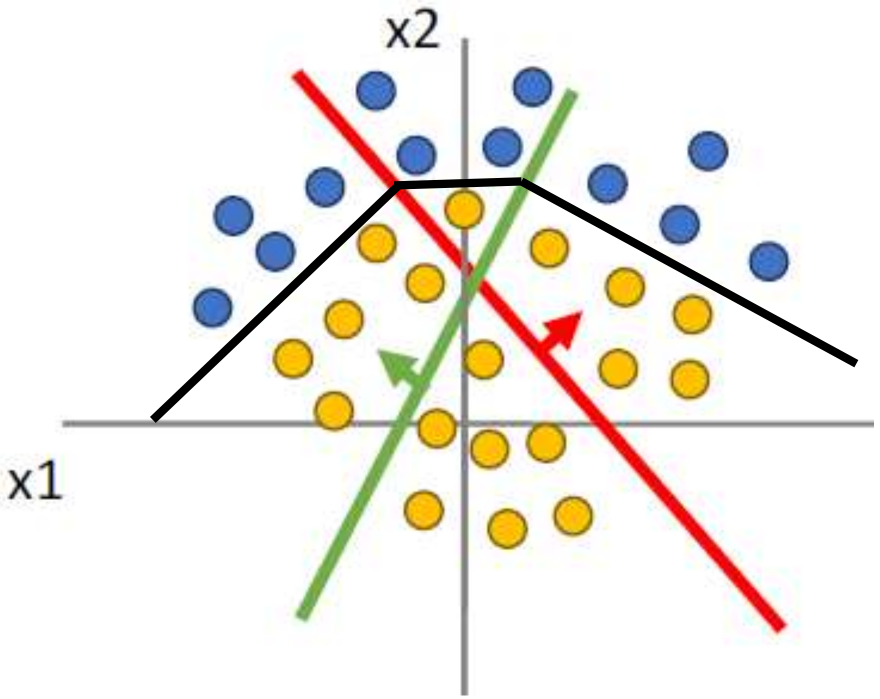




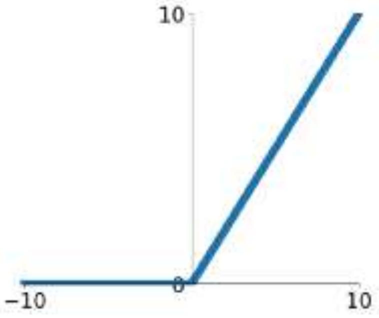
Points are linearly separable in features space!

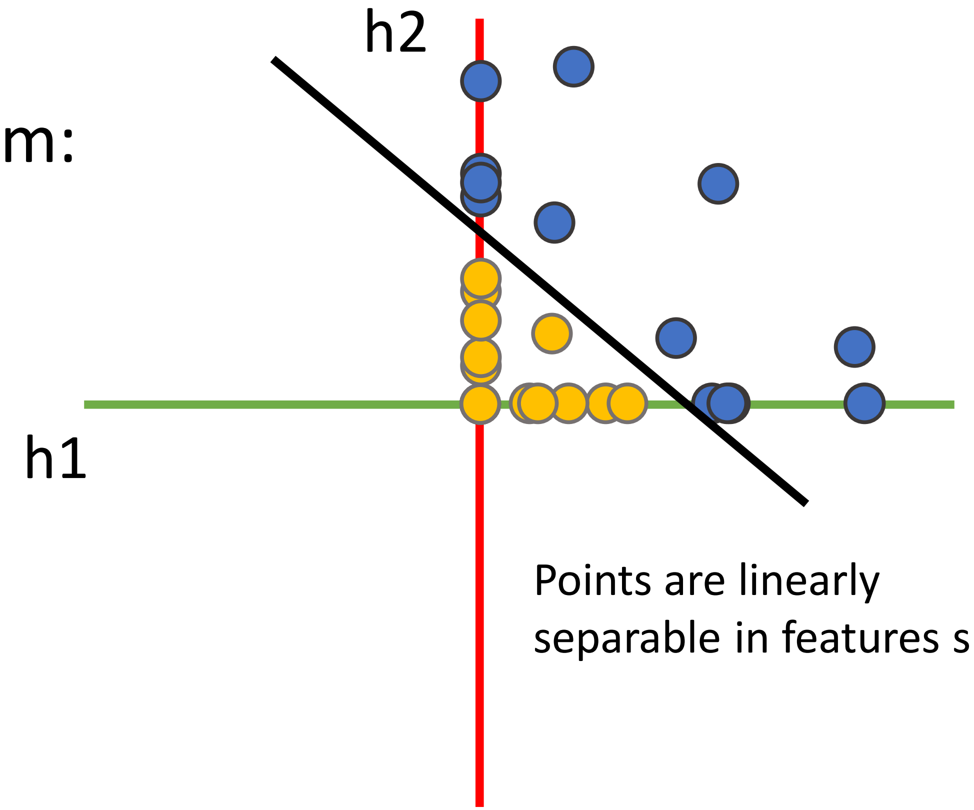
Space Warping

Consider a neural net hidden layer:
h = ReLU(Wx) = max(0, Wx)
Where x, h are both 2-dimensional



Feature transform:
h = ReLU(Wx)



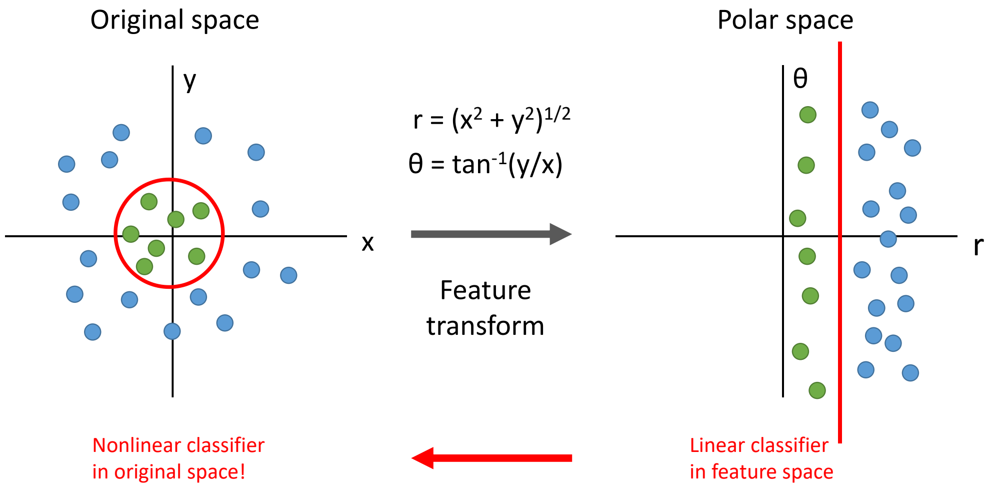


Points are linearly separable in features space!

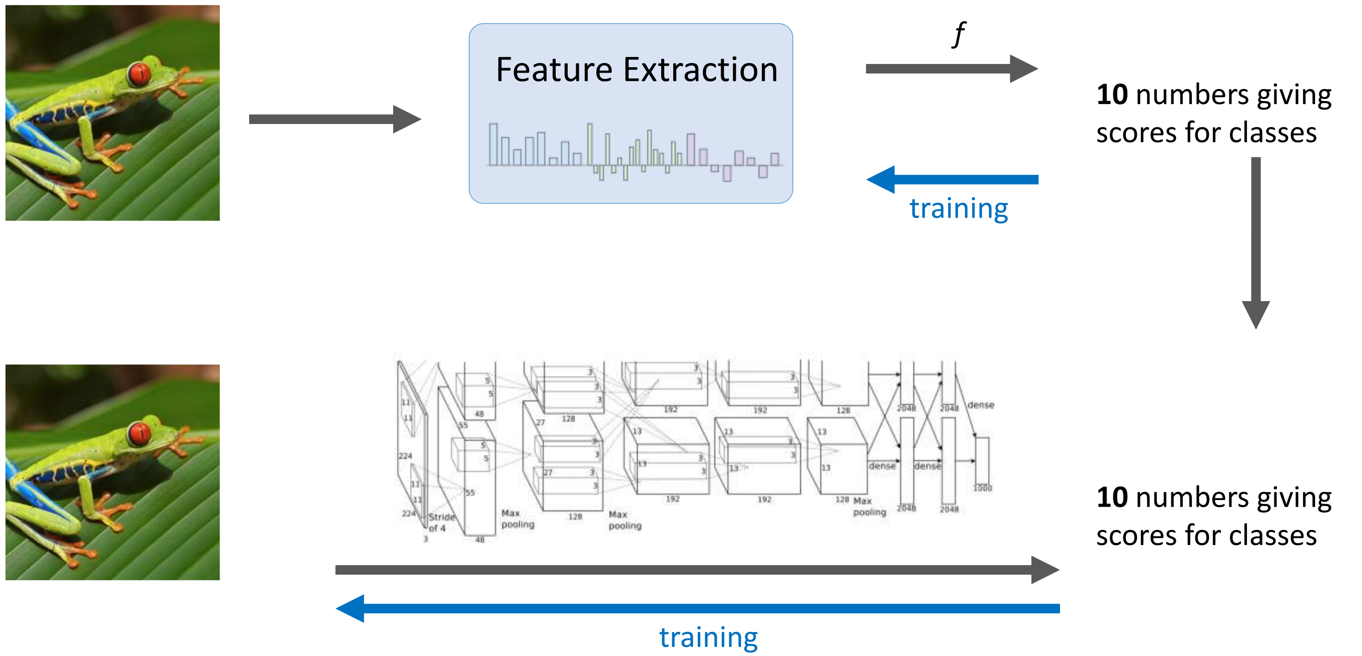
Linear classifier in feature space gives nonlinear classifier in original space

Summary

Feature transform + Linear classifer allows nonlinear decision boundaries



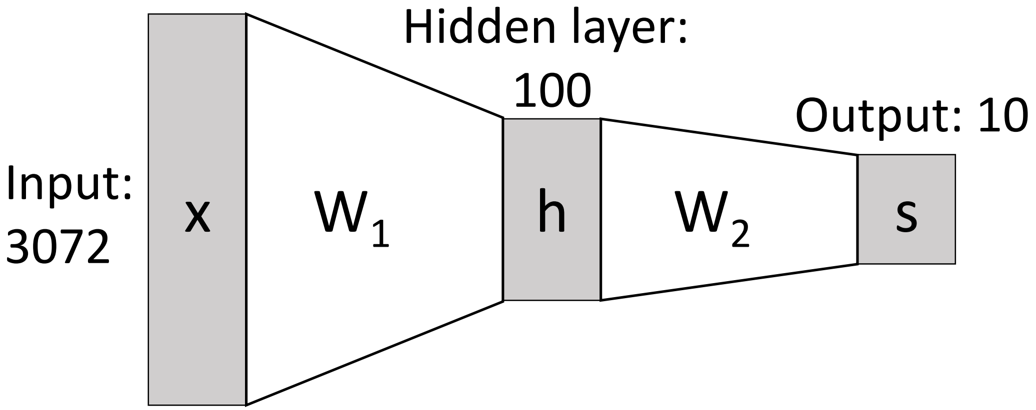
Neural Networks as learnable feature transforms



Summary

From linear classifers to
fully-connected networks

$$f = W_2 \max(0, W_1 x)$$



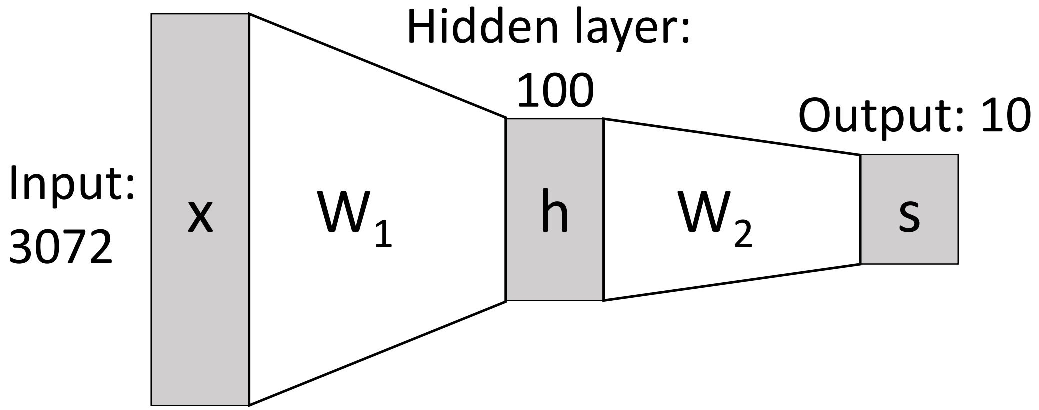
Linear classifier: One template per class

Neural networks: Many reusable templates

Summary

From linear classifers to
fully-connected networks

$$f = W_2 \max(0, W_1 x)$$



Neural networks loosely inspired by biological neurons but be careful with analogies