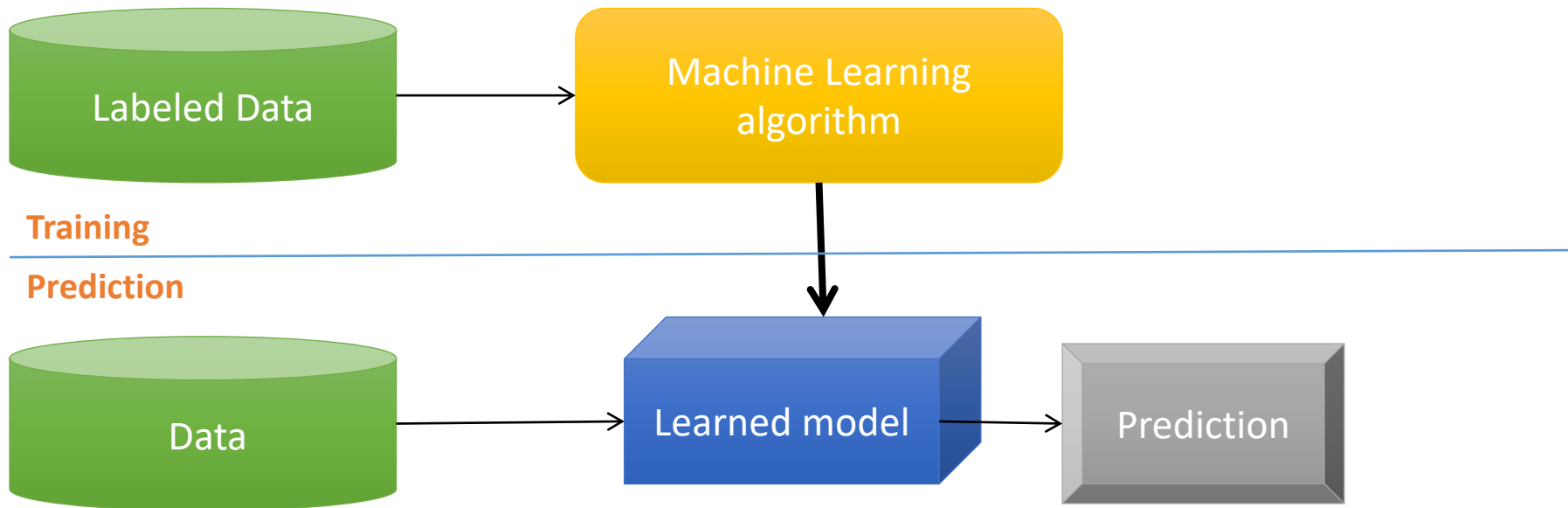


Deep Learning Tutorial

Courtesy of Hung-yi Lee

Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**



Methods that can learn from and make predictions on data

Types of Learning

Supervised: Learning with a **labeled training** set

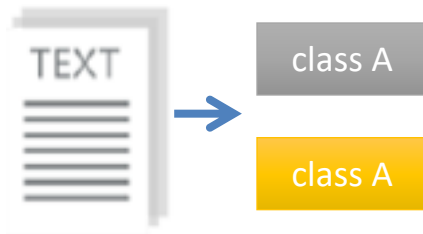
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled** data

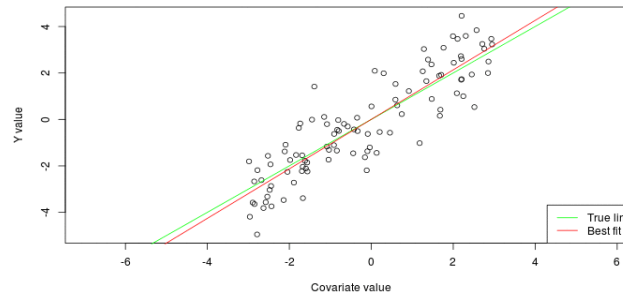
Example: *cluster* similar documents based on text

Reinforcement learning: learn to **act** based on **feedback/reward**

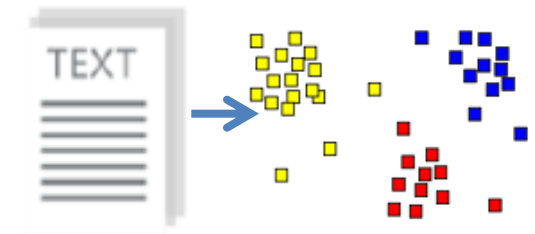
Example: learn to play Go, reward: *win or lose*



Classification



Regression



Clustering

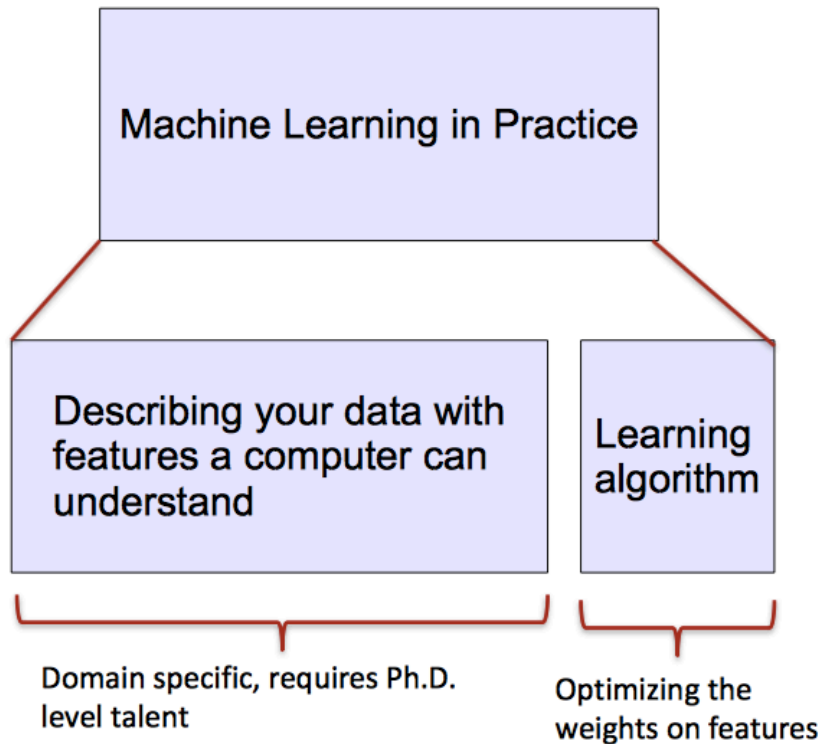
Anomaly Detection
Sequence labeling

...

ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**

ML becomes just **optimizing weights** to best make a final prediction



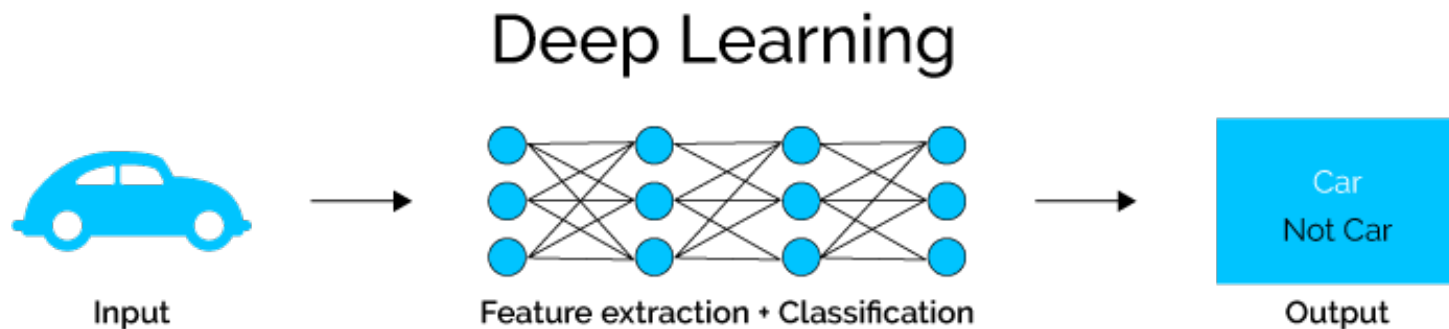
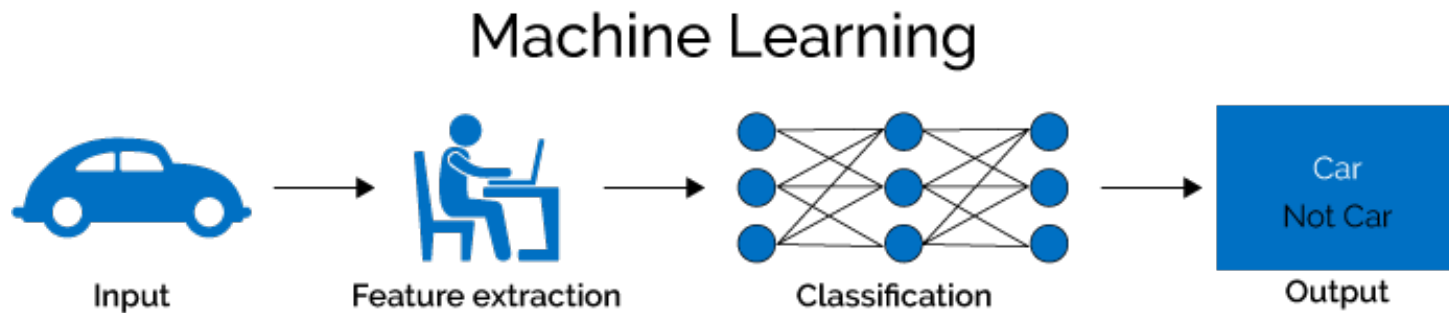
Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

What is Deep Learning (DL) ?

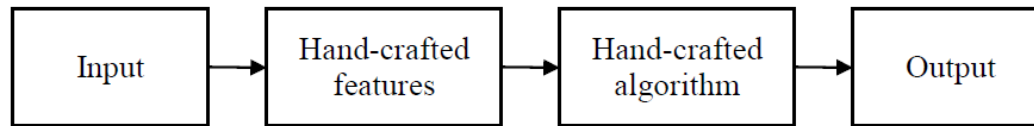
A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.

Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**

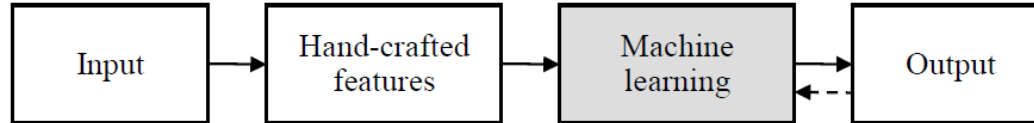
If you provide the system **tons of information**, it begins to understand it and respond in useful ways.



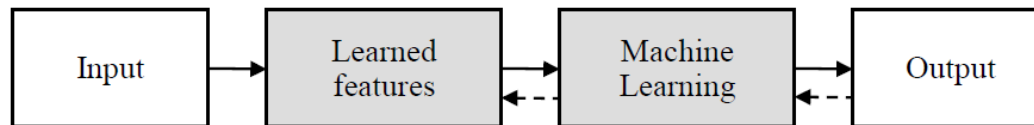
Traditional and deep learning



(a) Traditional vision pipeline



(b) Classic machine learning pipeline



(c) Deep learning pipeline

Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

In ~2010 DL started outperforming other ML techniques
first in speech and vision, then NLP

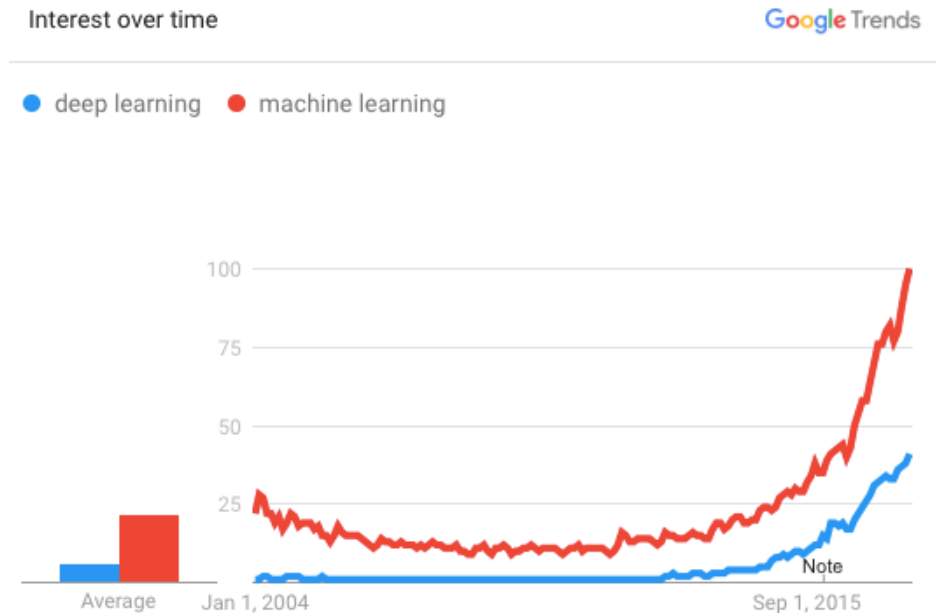


Image Classification: A core task in Computer Vision



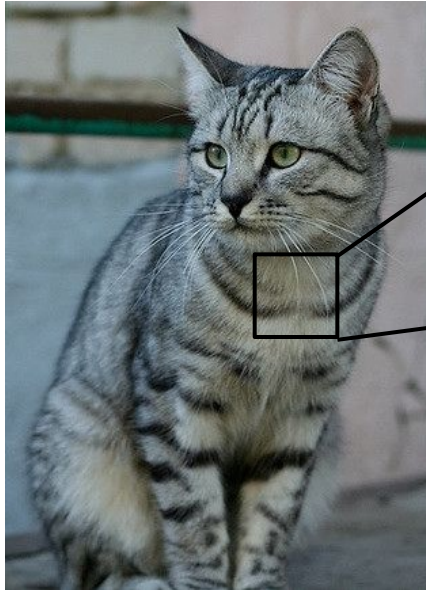
[This image](#) by [Nikita](#) is
licensed under [CC-BY 2.0](#)

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}



cat

The Problem: Semantic Gap



This image by Nikita.js
licensed under [CC-BY 2.0](https://creativecommons.org/licenses/by/2.0/)

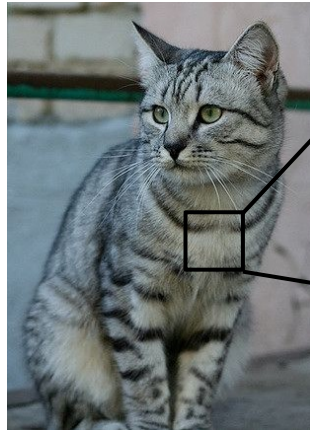
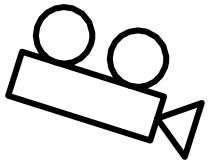
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]  
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]  
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]  
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]  
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]  
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]  
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]  
[ 89 93 90 97 106 147 131 118 113 114 113 109 106 95 77 60]  
[ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]  
[ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]  
[ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]  
[ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]  
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]  
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]  
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]  
[128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]  
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]  
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]  
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



[105	112	108	111	104	99	106	99	96	103	112	119	104	97	93	87]
[91	98	102	106	104	79	98	103	99	105	123	136	110	105	94	85]
[76	85	90	105	128	105	87	96	95	99	115	112	106	103	99	85]
[99	81	81	93	120	131	127	100	95	98	102	99	96	93	101	94]
[106	91	61	64	69	91	88	85	101	107	109	98	75	84	96	95]
[114	100	05	55	55	69	64	54	64	87	112	125	98	74	84	91]
[133	137	147	103	65	81	80	65	52	54	74	84	102	93	85	82]
[128	137	144	140	109	95	86	70	62	65	63	63	60	73	86	101]
[125	133	148	137	119	121	117	94	65	79	80	65	54	64	72	90]
[127	125	131	147	133	127	126	131	111	96	89	75	61	64	72	84]
[115	114	109	123	150	148	131	118	113	109	100	92	74	65	72	78]
[89	93	98	97	108	147	131	118	113	114	113	100	106	95	77	80]
[63	77	86	81	77	79	102	123	117	115	117	125	125	130	115	87]
[62	65	82	89	78	71	80	101	124	126	119	101	107	114	131	119]
[63	65	75	88	69	71	62	81	128	138	135	105	81	98	110	118]
[87	65	71	97	186	95	69	65	76	130	126	107	92	94	105	112]
[118	97	82	86	117	123	116	66	41	51	95	93	89	95	102	107]
[164	146	112	80	82	120	124	104	76	48	45	66	88	101	102	109]
[157	170	157	120	63	86	114	132	112	97	69	55	70	82	99	94]
[130	128	134	161	139	100	109	118	121	134	114	87	65	53	69	86]
[120	112	96	117	150	144	120	115	104	107	102	93	87	81	72	79]
[123	107	96	86	83	112	153	149	122	109	104	75	80	107	112	99]
[122	121	102	80	82	86	94	117	145	148	153	102	58	78	92	107]
[122	164	148	103	71	56	78	83	93	103	119	139	102	61	69	84]

All pixels change when the camera moves!

Challenges: Illumination



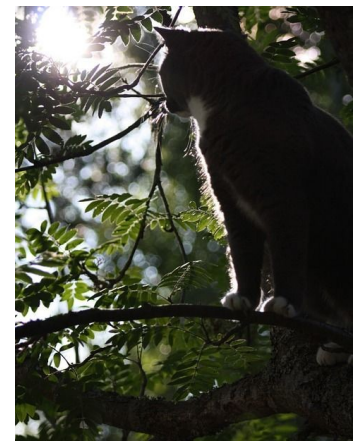
[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain

Challenges: Deformation



This image by [Umberto Salvagnin](#) is licensed under [CC-BY 2.0](#)



This image by [Umberto Salvagnin](#) is licensed under [CC-BY 2.0](#)



This image by [sare bear](#) is licensed under [CC-BY 2.0](#)



This image by [Tom Thai](#) is licensed under [CC-BY 2.0](#)

1
3

Challenges: Occlusion



This image is [CC0 1.0](#) public domain



This image is [CC0 1.0](#) public domain



This image by [jonsson](#) is licensed under [CC-BY 2.0](#)

1
4

Challenges: Background Clutter



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain

Challenges: Intra-class variation



[This image](#) is [CC0 1.0](#) public domain

Linear Classification

1
1
3
7

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 $32 \times 32 \times 3$ numbers
(3072 numbers total)



10 numbers giving
class scores

↑
 \mathbf{W}

parameters
or weights

01
1
in
9
8

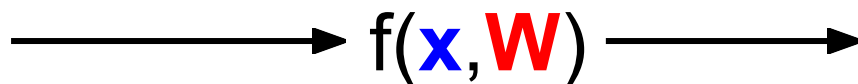
Parametric Approach: Linear Classifier

Image



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

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



$f(x, W)$

10 numbers giving
class scores



W

parameters
or weights

Parametric Approach: Linear Classifier

Image



Array of $32 \times 32 \times 3$ numbers
(3072 numbers total)

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

10×1 10×3072



10 numbers giving
class scores



W

parameters
or weights

Parametric Approach: Linear Classifier

Image

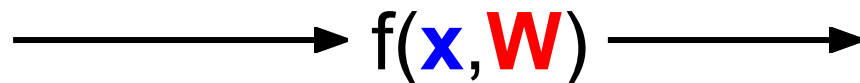


Array of $32 \times 32 \times 3$ numbers
(3072 numbers total)

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

The equation is annotated with dimensions: W is 10×3072 , x is 3072×1 , and b is 10×1 .

10×1 10×3072



10 numbers giving
class scores

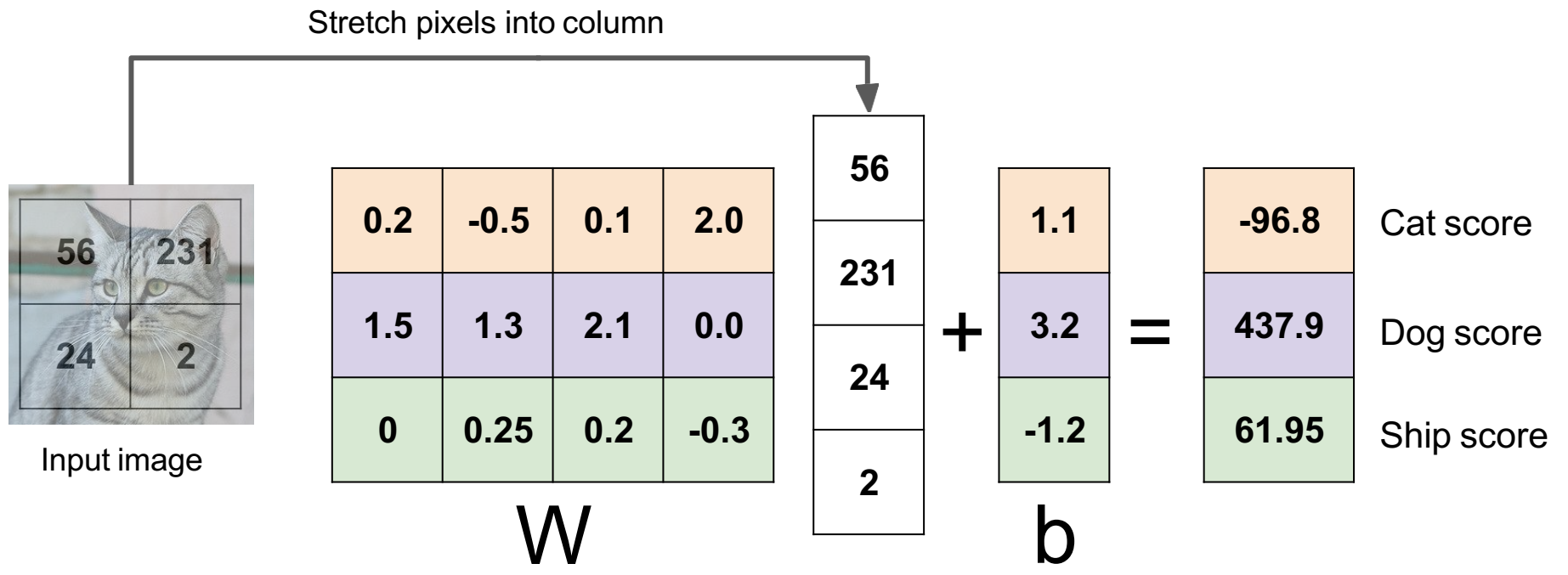


W

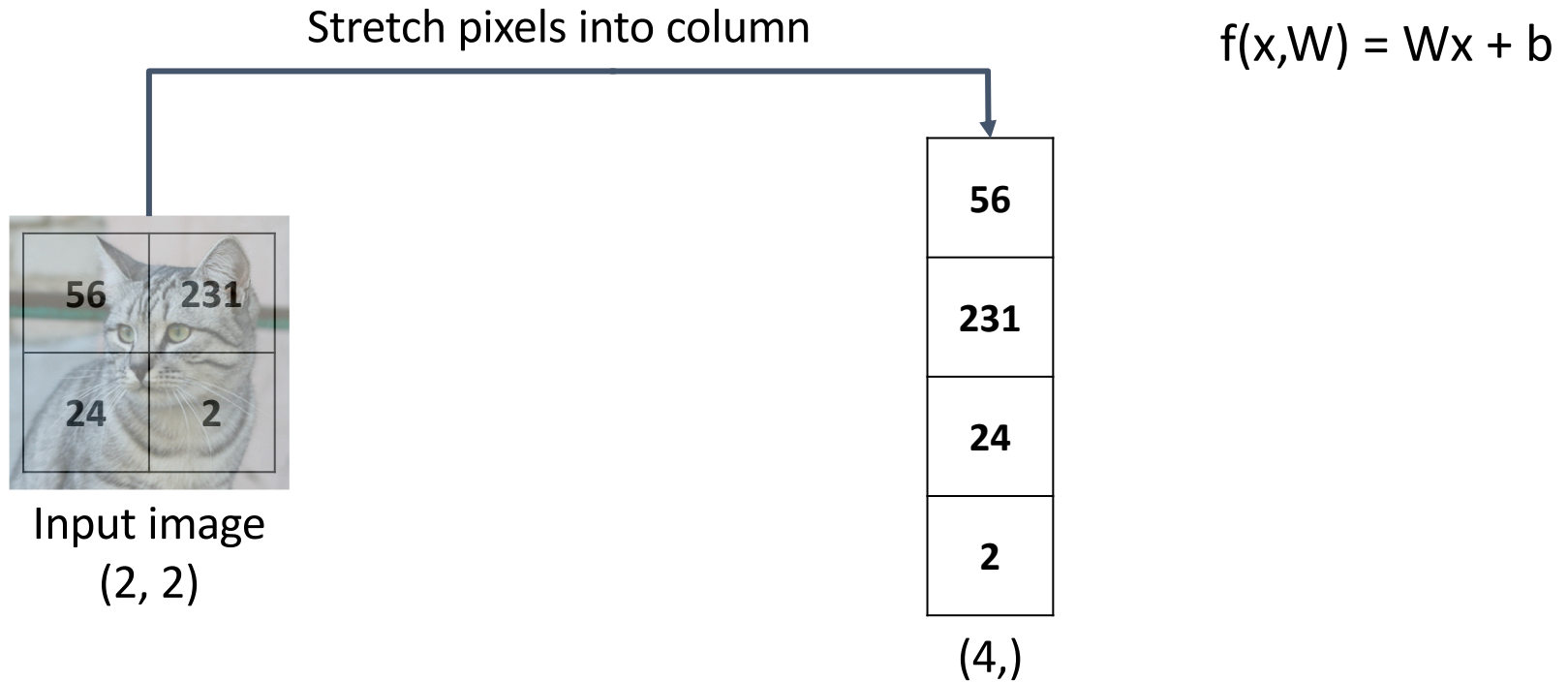
parameters
or weights

1
2
2
2

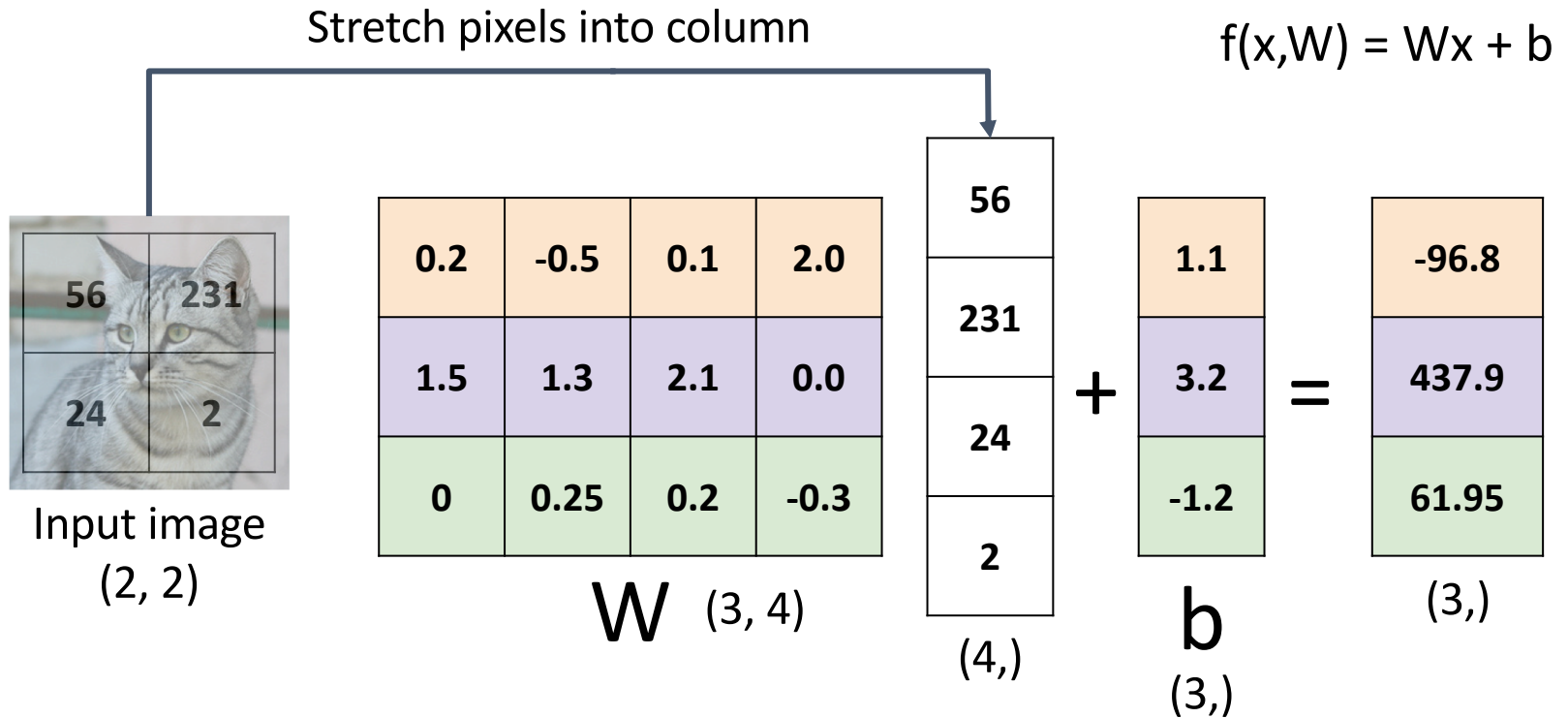
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



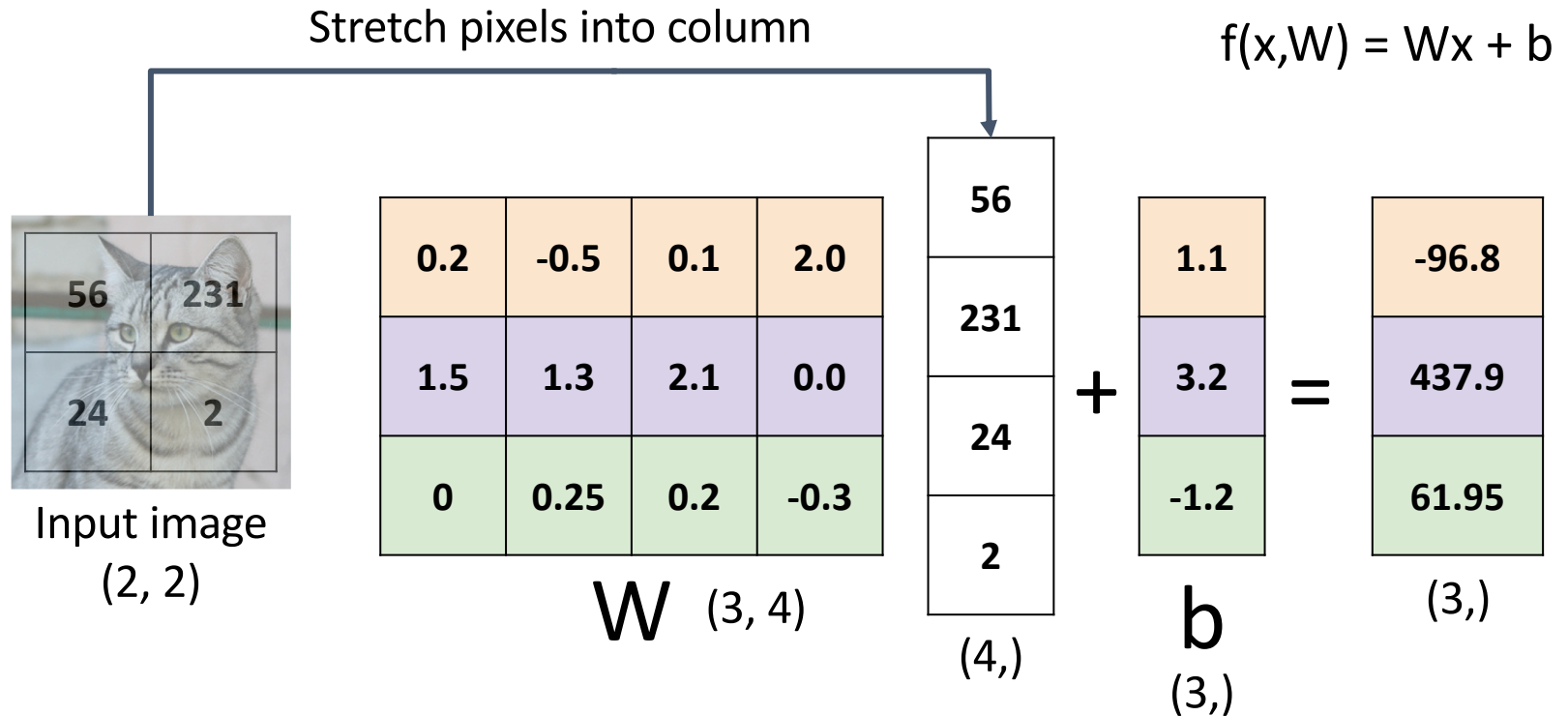
Example for 2x2 image, 3 classes (cat/dog/ship)



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



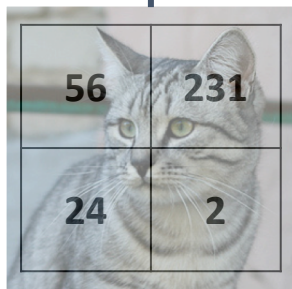
Linear Classifier: Algebraic Viewpoint



Linear Classifier: Bias Trick

Add extra one to data vector;
bias is absorbed into last
column of weight matrix

Stretch pixels into column



Input image
(2, 2)

0.2	-0.5	0.1	2.0	1.1
1.5	1.3	2.1	0.0	3.2
0	0.25	0.2	-0.3	-1.2

W (3, 5)

56
231
24
2
1

=

-96.8
437.9
61.95

(3,)

Linear Classifier: Predictions are Linear!

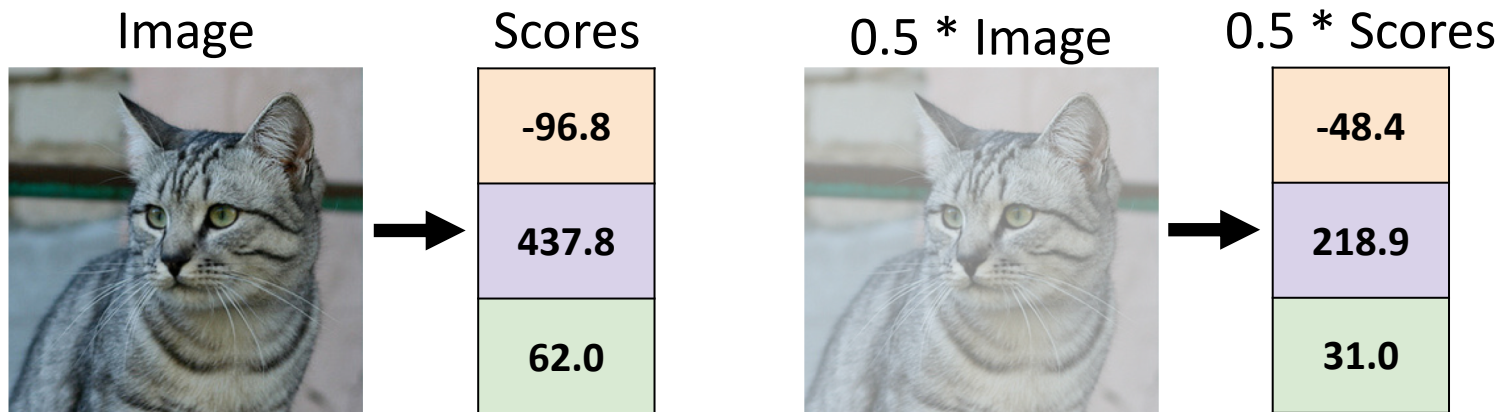
$$f(x, W) = Wx \quad (\text{ignore bias})$$

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

Linear Classifier: Predictions are Linear!

$$f(x, W) = Wx \quad (\text{ignore bias})$$

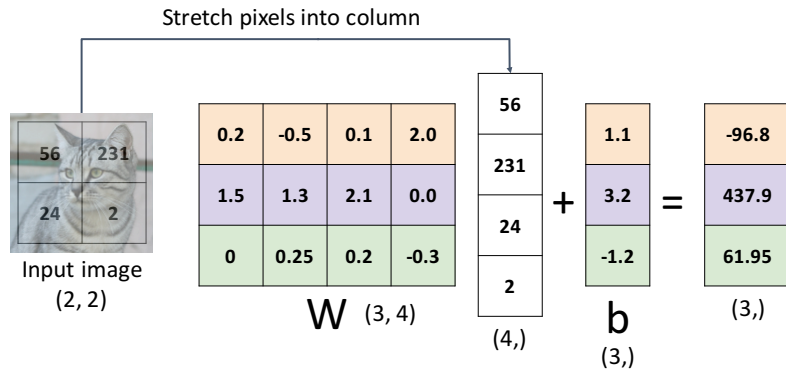
$$f(cx, W) = W(cx) = c * f(x, W)$$



Interpreting a Linear Classifier

Algebraic Viewpoint

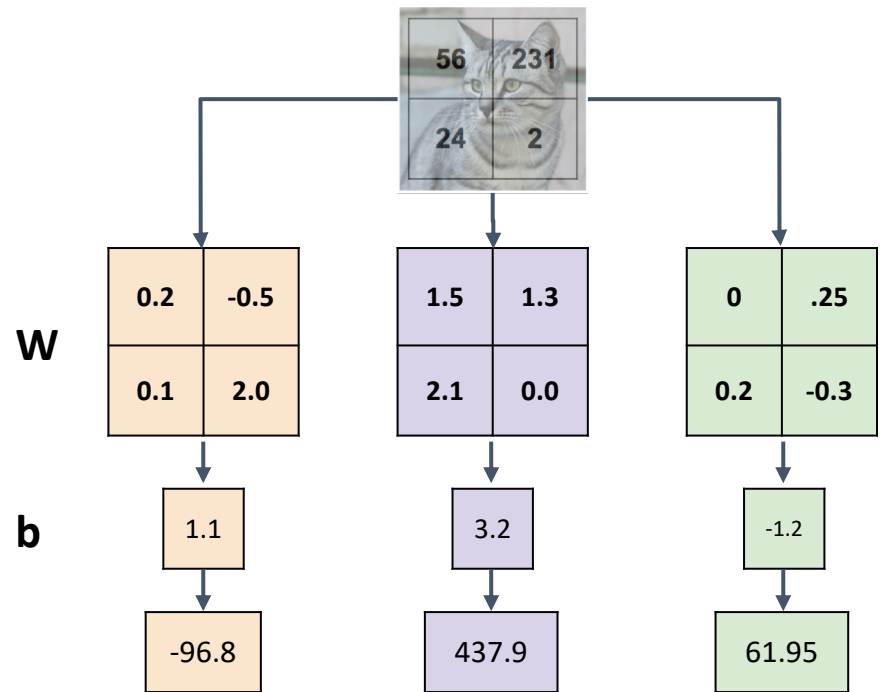
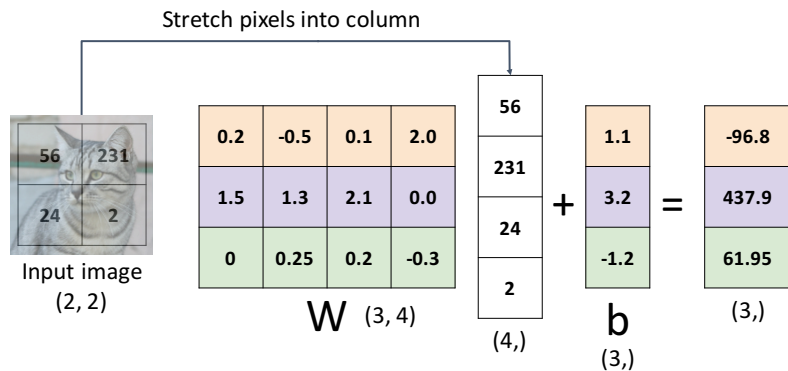
$$f(x,W) = Wx + b$$



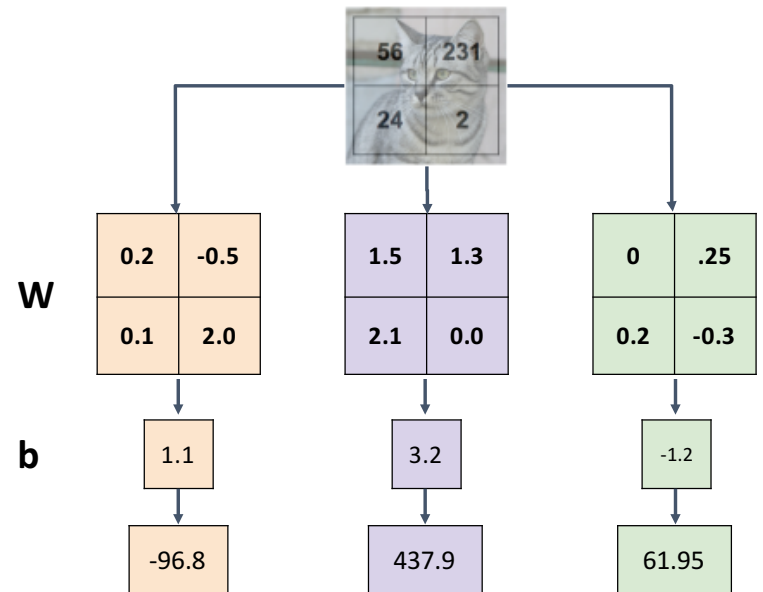
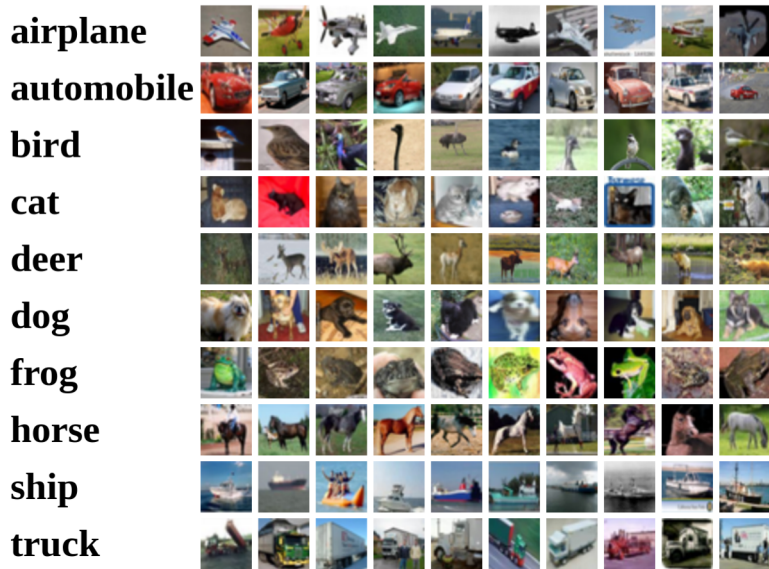
Interpreting a Linear Classifier

Algebraic Viewpoint

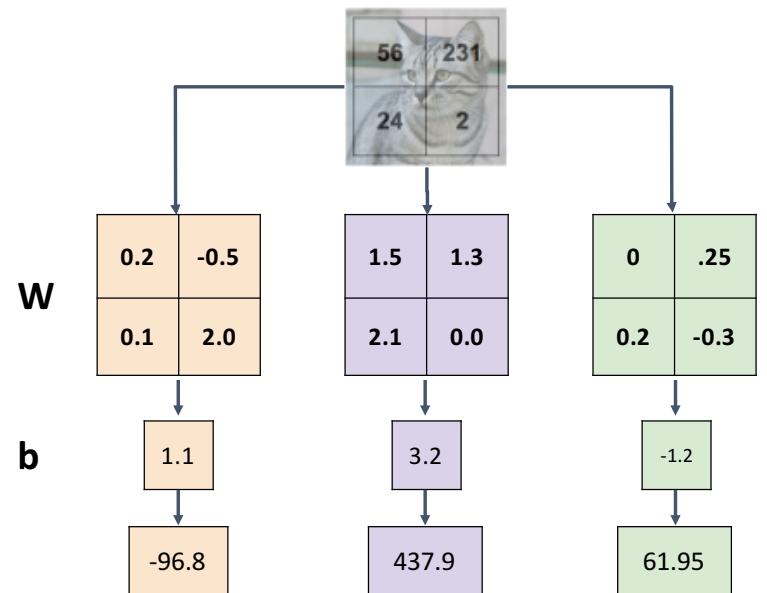
$$f(x, W) = Wx + b$$



Interpreting an Linear Classifier

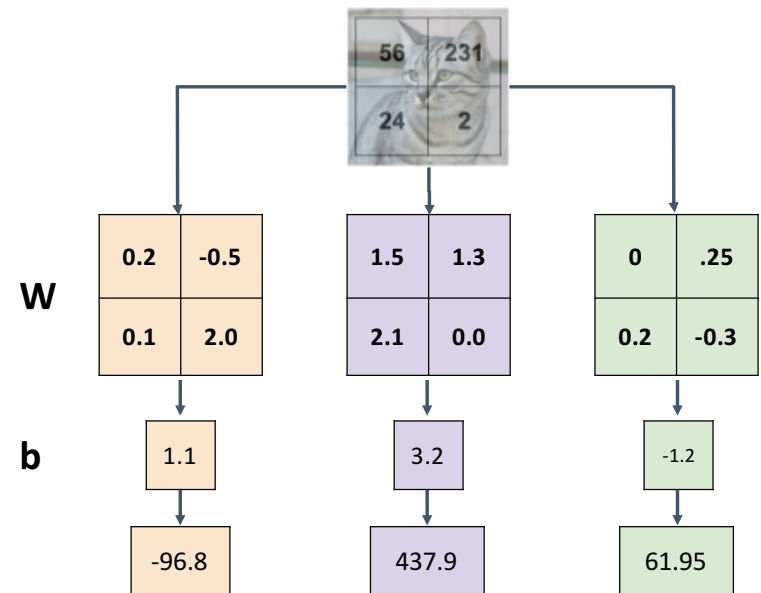


Interpreting an Linear Classifier: Visual Viewpoint



Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one
“template” per
category

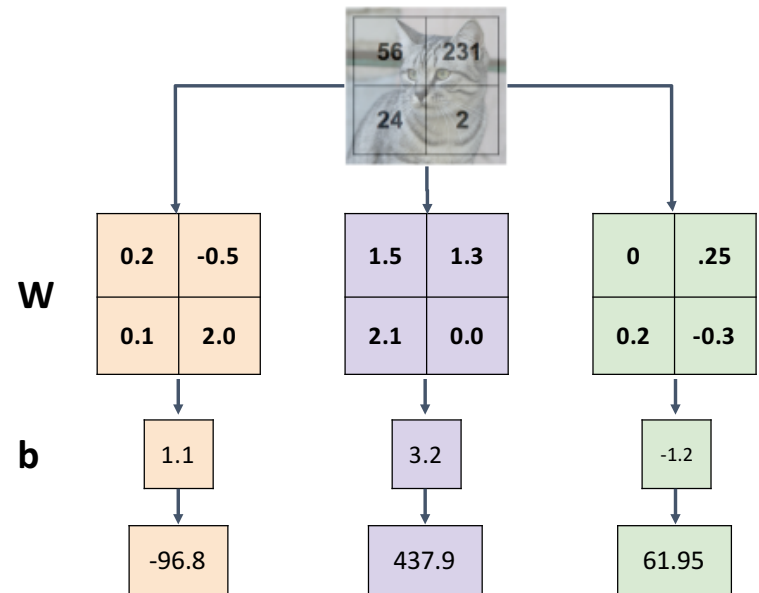


Interpreting an Linear Classifier: Visual Viewpoint

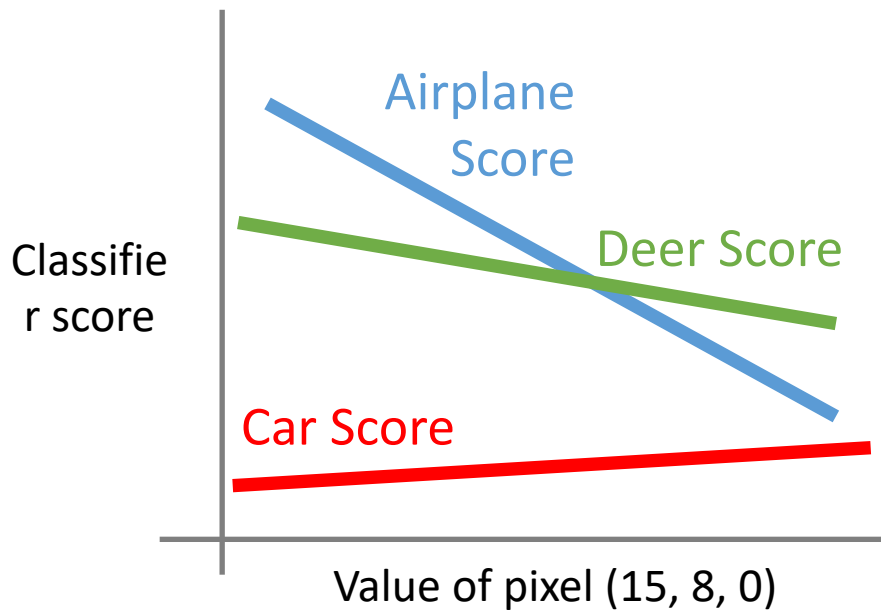
Linear classifier has one
“template” per
category

A single template cannot capture
multiple modes of the data

e.g. horse template has 2 heads!



Interpreting a Linear Classifier: Geometric Viewpoint

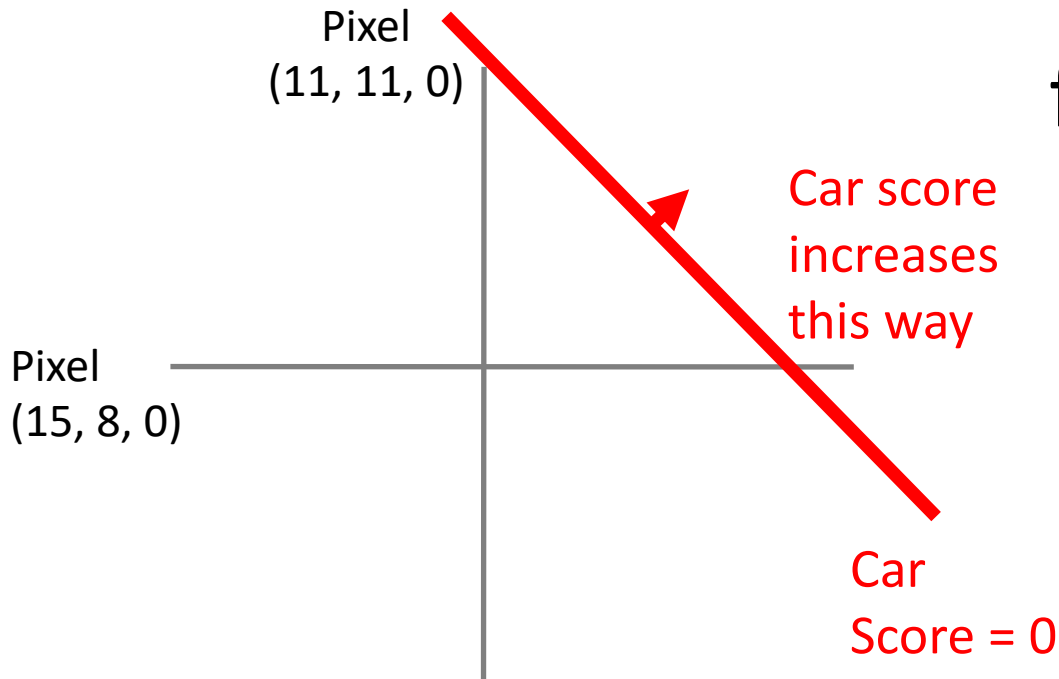


$$f(x, W) = Wx + b$$



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

Interpreting a Linear Classifier: Geometric Viewpoint

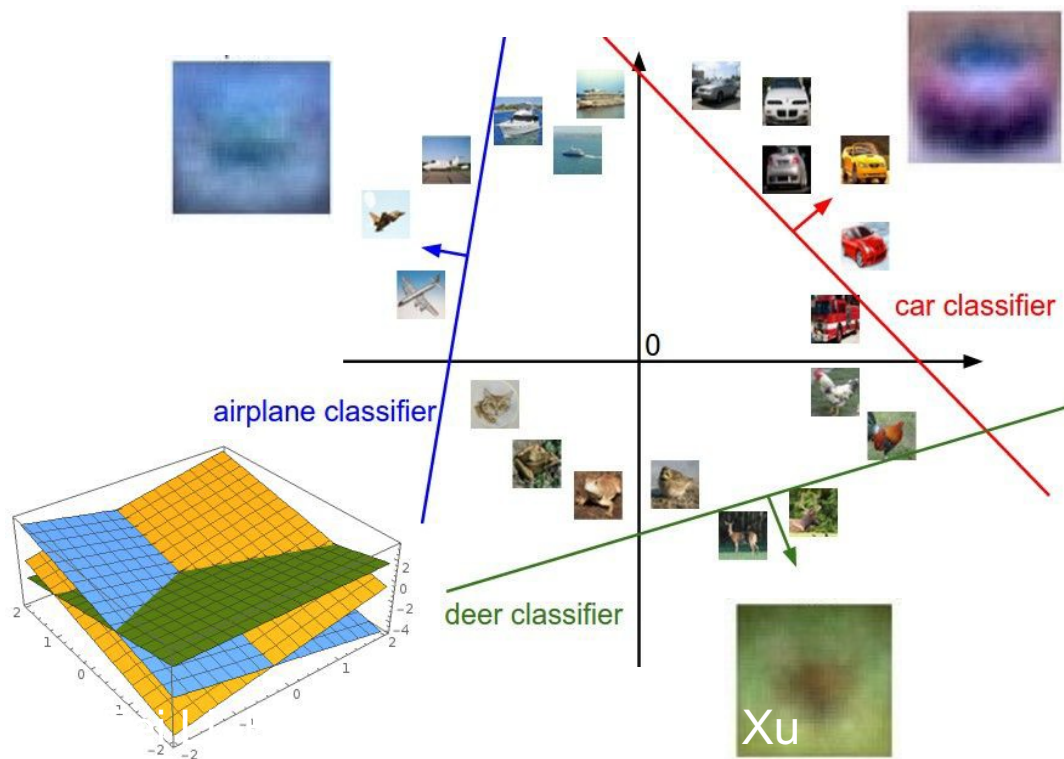


$$f(x, W) = Wx + b$$



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

Interpreting a Linear Classifier: Geometric Viewpoint



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



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

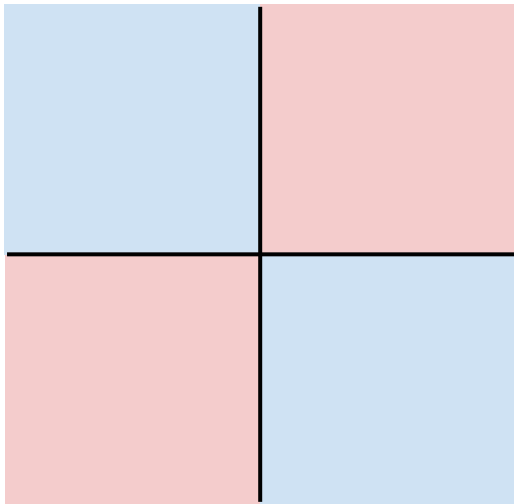
Hard Cases for a Linear Classifier

Class 1:

First and third quadrants

Class 2:

Second and fourth quadrants

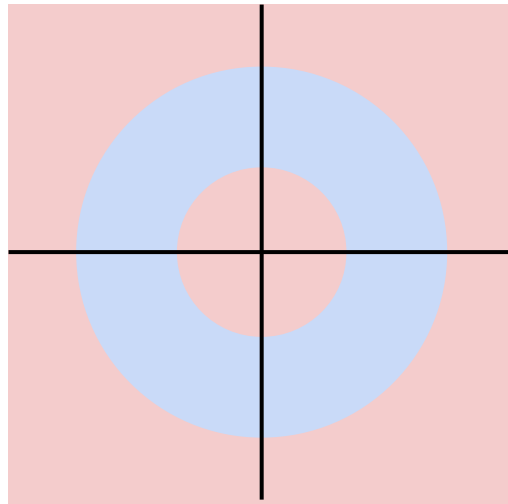


Class 1:

$1 \leq L2 \text{ norm} \leq 2$

Class 2:

Everything else

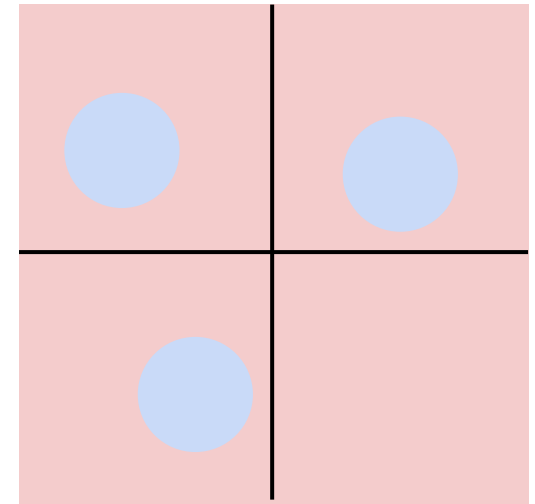


Class 1:

Three modes

Class 2:

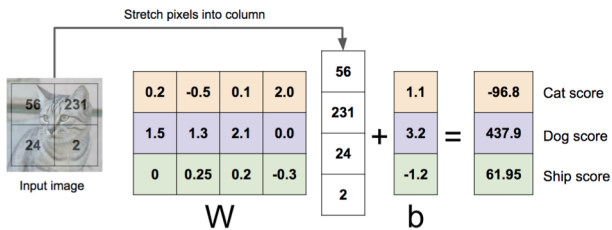
Everything else



Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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



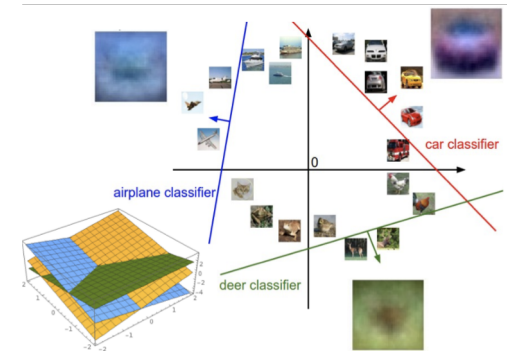
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



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 ?

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#); [Car image](#) is [CC0 1.0](#) public domain; [Frog image](#) is in the public domain

Choosing a good 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)

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

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 (integer) label

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 (integer) label

Loss for a single example is

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

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 (integer) label

Loss for a single example is

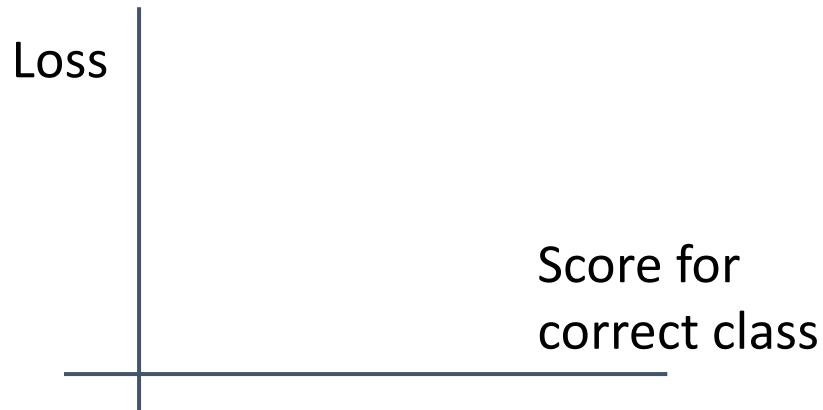
$$L_i(f(x_i, W), y_i)$$

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

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

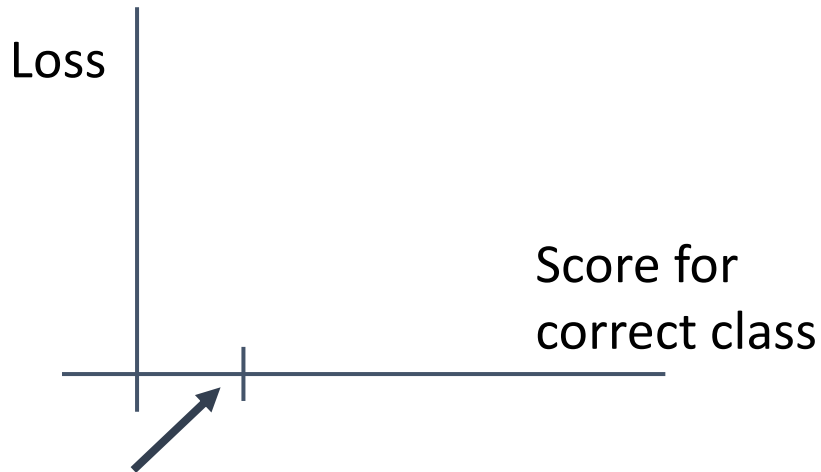
Multiclass SVM Loss

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



Multiclass SVM Loss

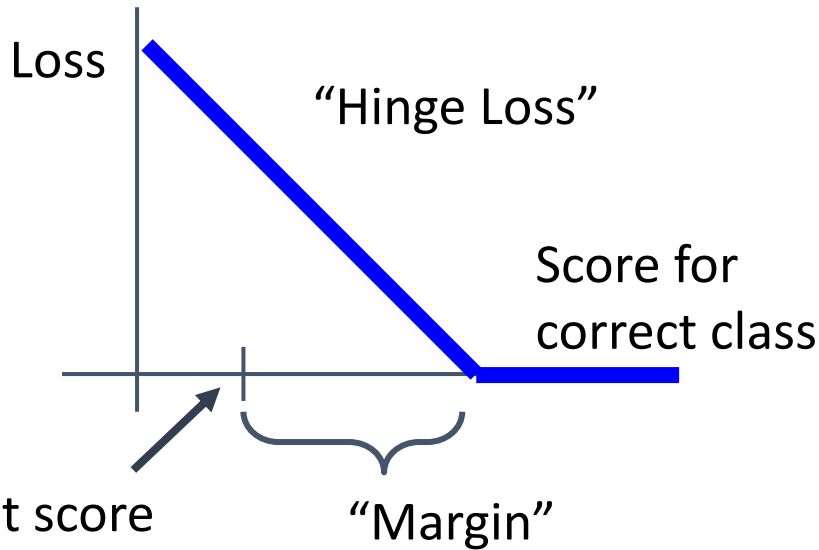
”The score of the correct class should be higher than all the other scores”



Highest score
among other
classes

Multiclass SVM Loss

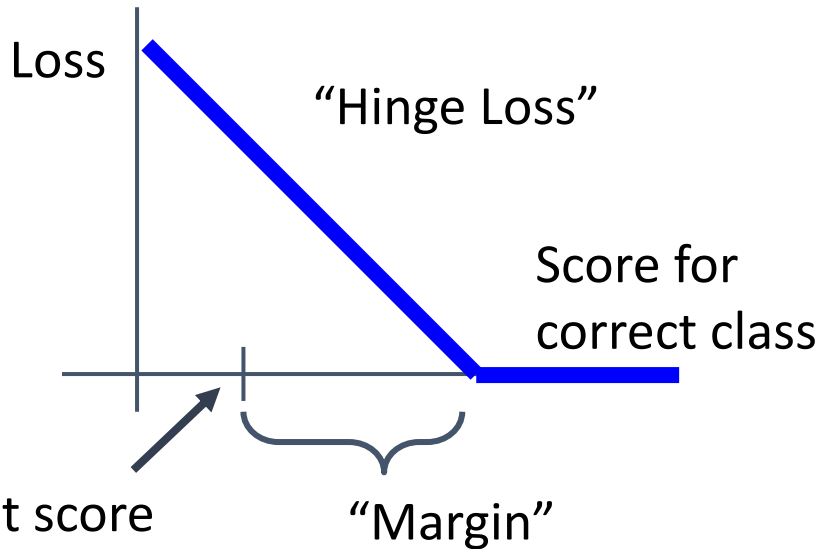
"The score of the correct class should be higher than all the other scores"



Highest score among other classes

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:

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

Regularization: Beyond Training Error

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


Data loss: Model predictions should match training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Cutout, Mixup, Stochastic depth, etc...

Regularization: Beyond Training Error

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

λ = regularization strength (hyperparameter)

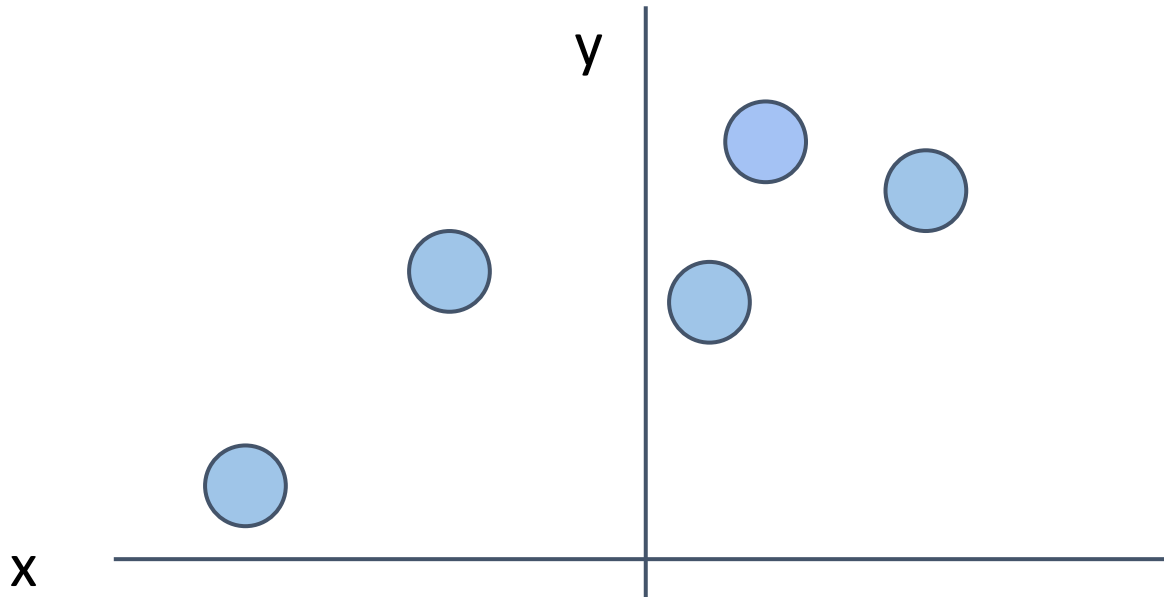
Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

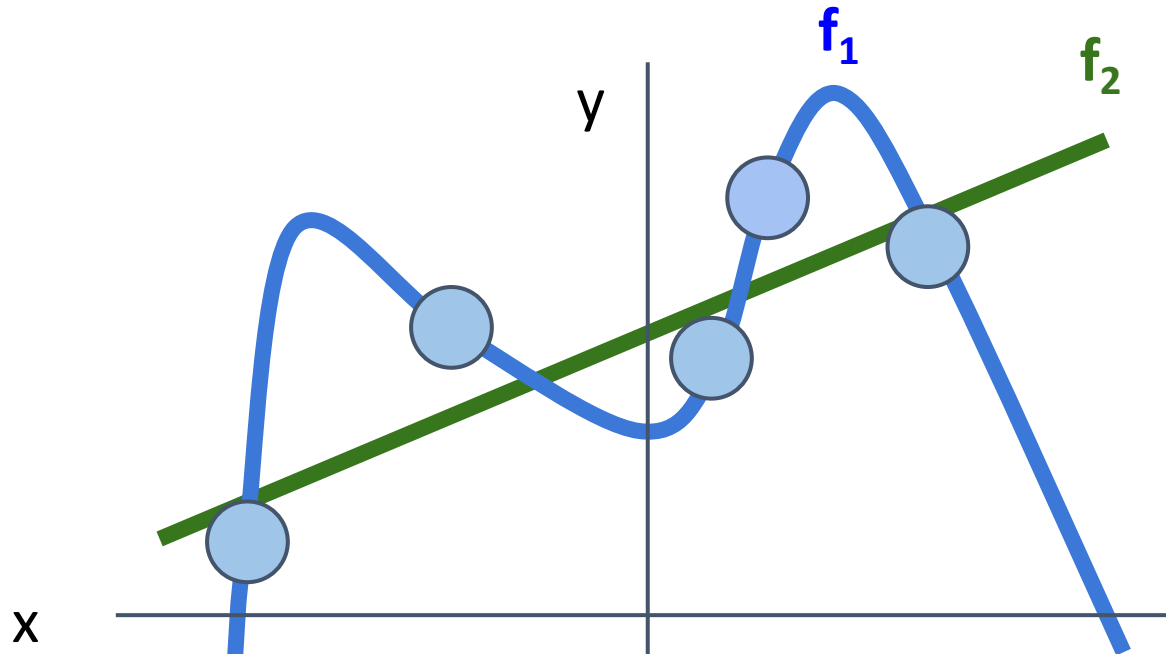
Purpose of Regularization:

- Express preferences in among models beyond "minimize training error"
- Avoid **overfitting**: Prefer simple models that generalize better
- Improve optimization by adding curvature

Regularization: Prefer Simpler Models

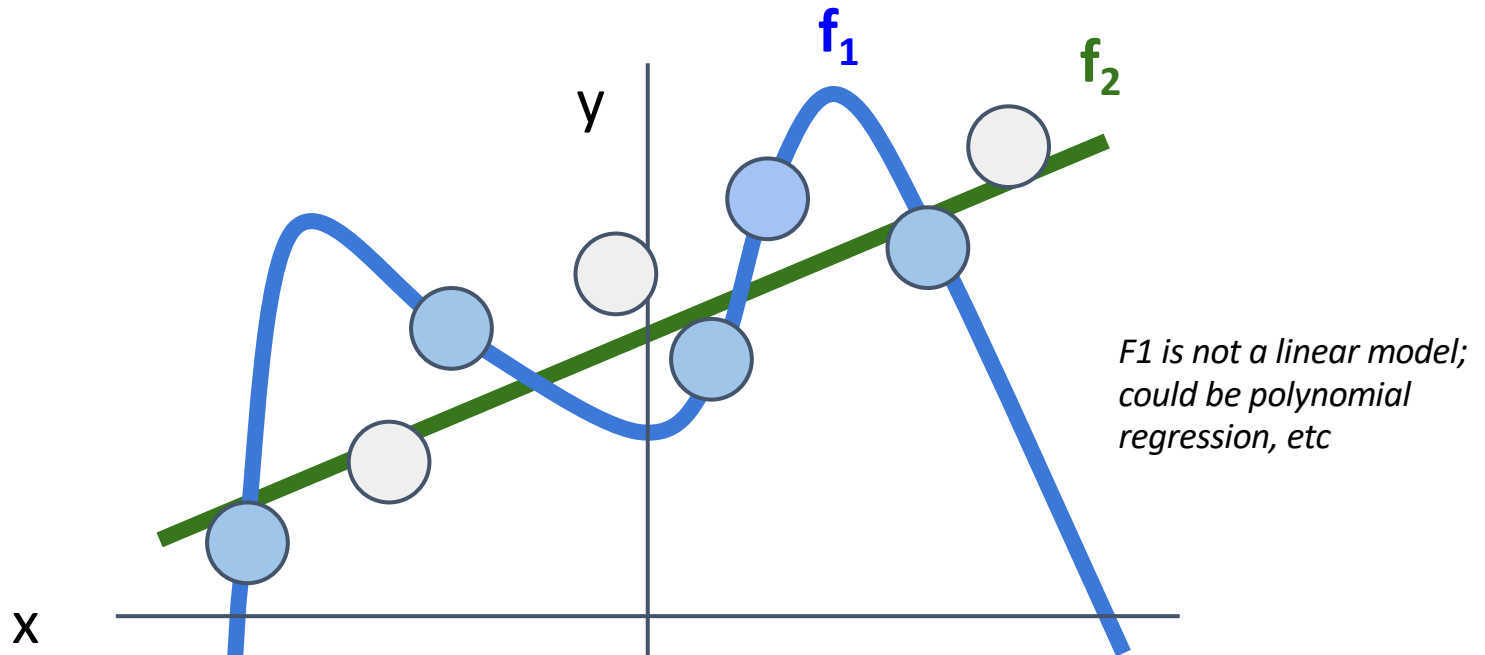


Regularization: Prefer Simpler Models



The model f_1 fits the training data perfectly
The model f_2 has training error, but is simpler

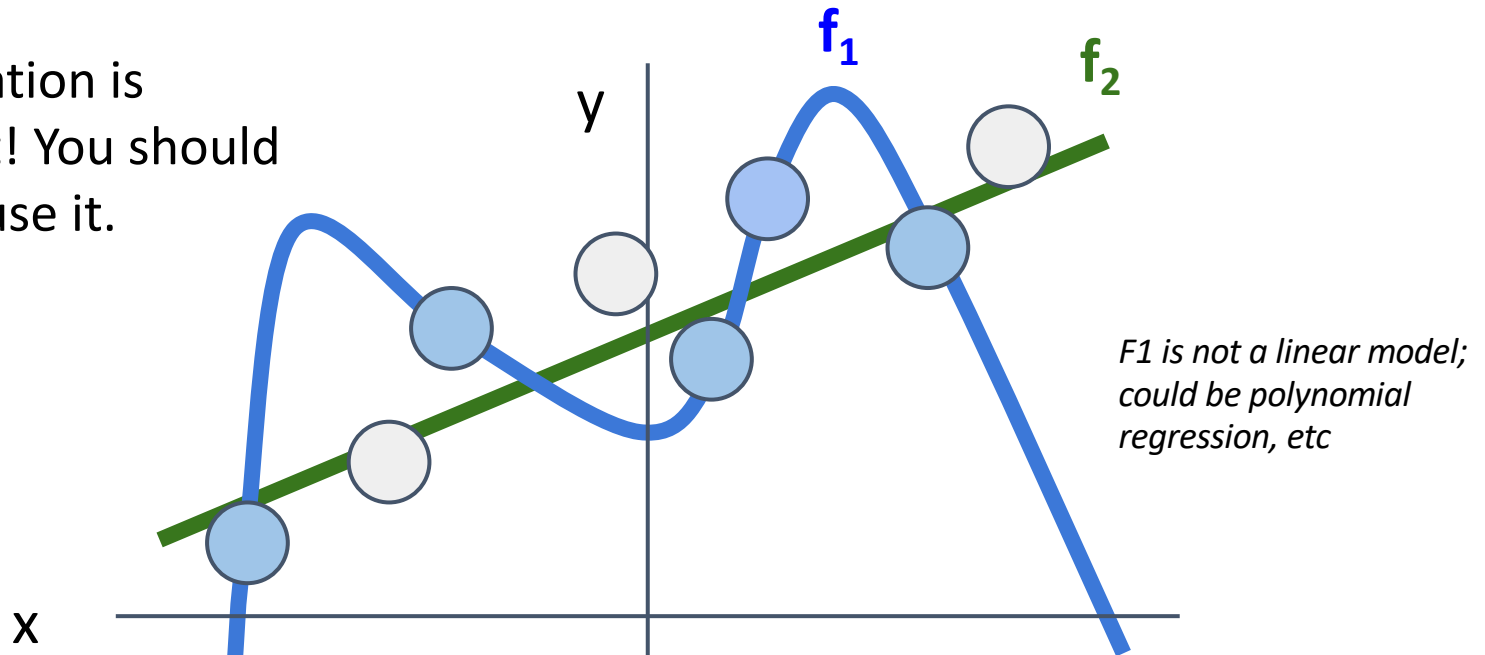
Regularization: Prefer Simpler Models



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Regularization: Prefer Simpler Models

Regularization is important! You should (usually) use it.



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



cat **3.2**

car 5.1

frog -1.7

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

cat **3.2**

car 5.1

frog -1.7

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

cat	3.2
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

cat	3.2	24.5
car	5.1	164.
frog	-1.7	0

Unnormalized log-probabilities / logits

unnormalized probabilities

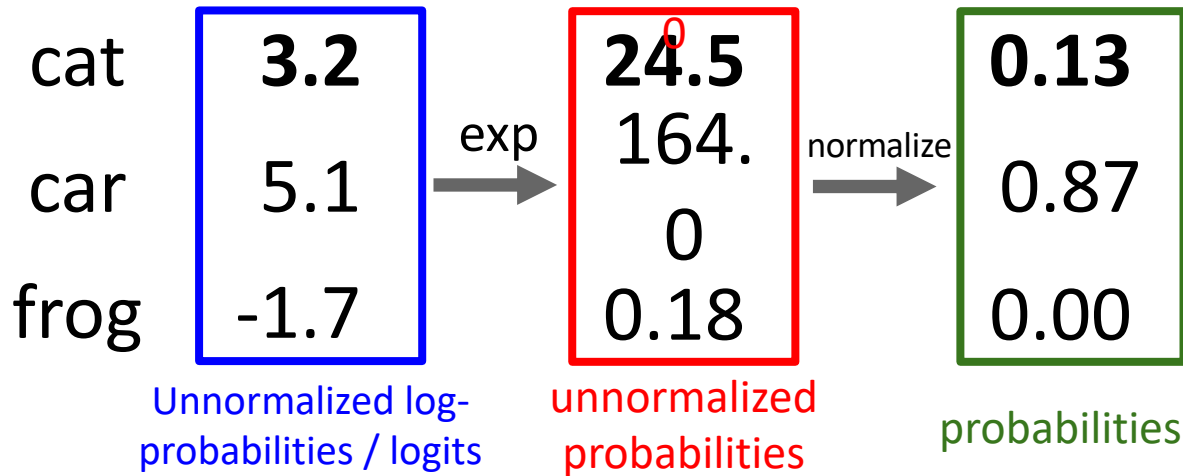
Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$



Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

cat	3.2
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

exp →

cat	24.5
car	164.
frog	0.18

unnormalized probabilities

normalize →

cat	0.13
car	0.87
frog	0.00

probabilities

$$L_i = -\log(0.13) = 2.04$$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

cat	3.2
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

exp

cat	24.5
car	164.
frog	0
	0.18

unnormalized probabilities

normalize

cat	0.13
car	0.87
frog	0.00

probabilities

$$L_i = -\log(0.13) = 2.04$$

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



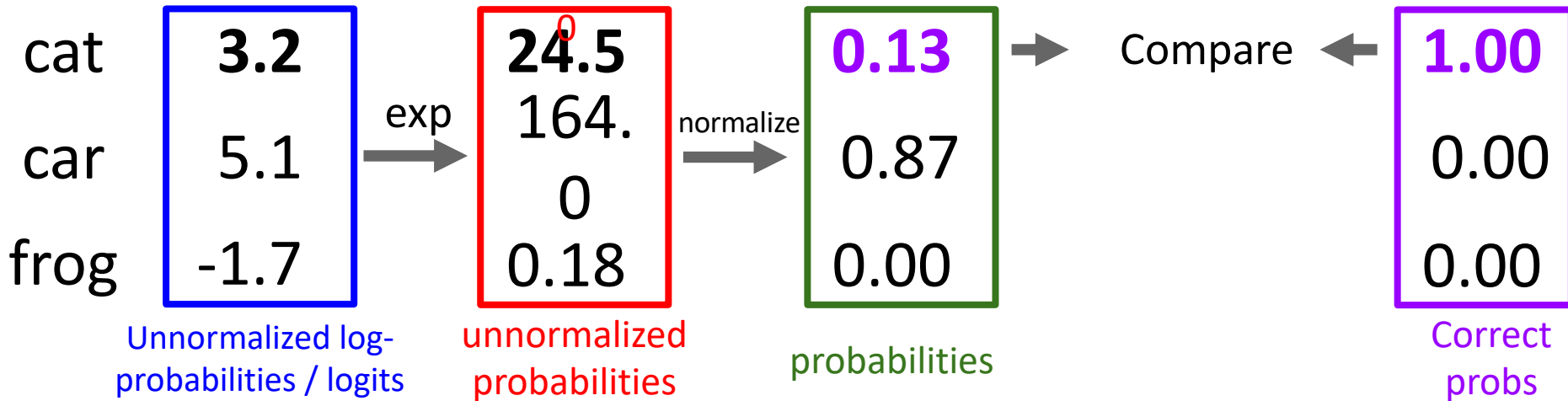
$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$



Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



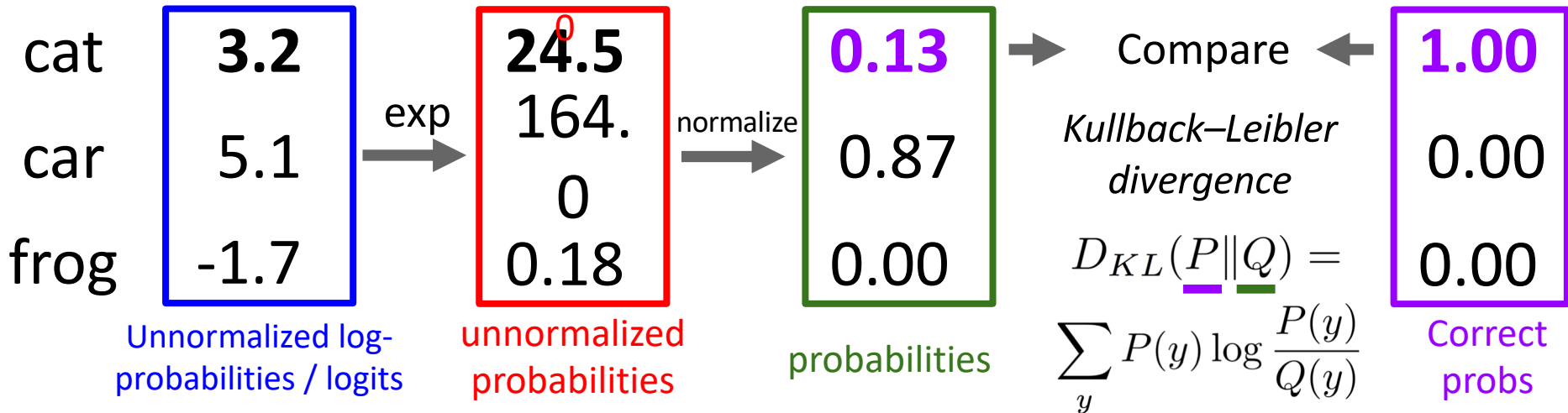
$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$



Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



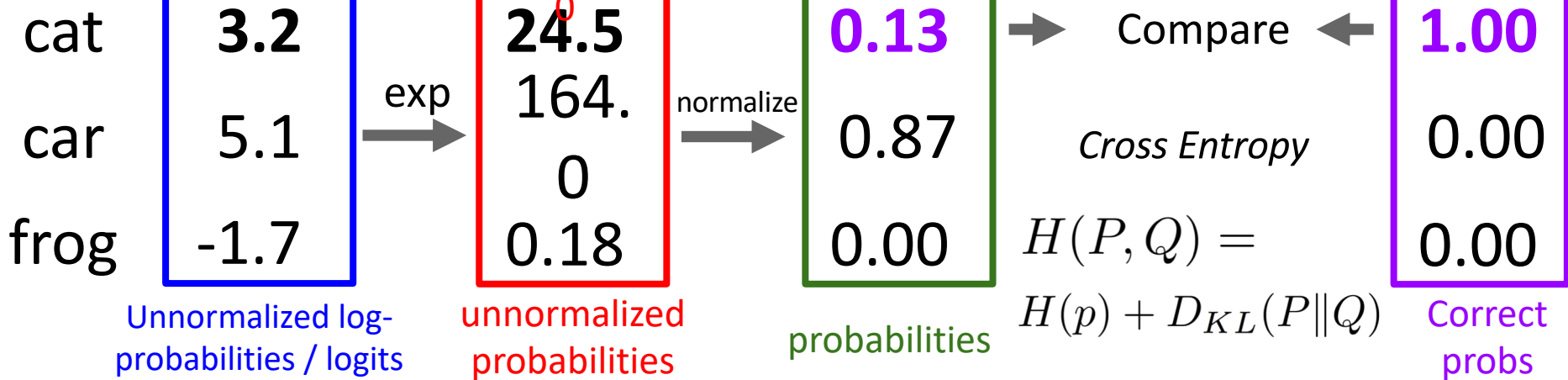
$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Probabilities must be ≥ 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$



Cross-Entropy Loss (Multinomial Logistic Regression)



cat **3.2**
car 5.1
frog -1.7

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Maximize probability of correct class

$$L_i = -\log P(Y = y_i | X = x_i)$$

Putting it all together:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Maximize probability of correct class

Putting it all together:

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

Q: What is the min / max possible loss L_i ?

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Maximize probability of correct class

Putting it all together:

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

Q: What is the min / max possible loss L_i ?

A: Min 0, max +infinity

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Maximize probability of correct class

Putting it all together:

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

Q: If all scores are small random values, what is the loss?

Cross-Entropy Loss (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**



$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{Softmax function}$$

Maximize probability of correct class

Putting it all together:

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

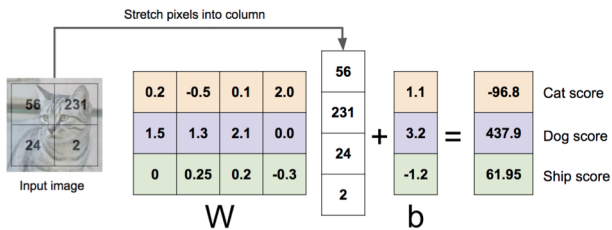
Q: If all scores are small random values, what is the loss?

A: $-\log(1/C)$
 $\log(10) \approx 2.3$

Recap: Three ways to think about linear classifiers

Algebraic Viewpoint

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



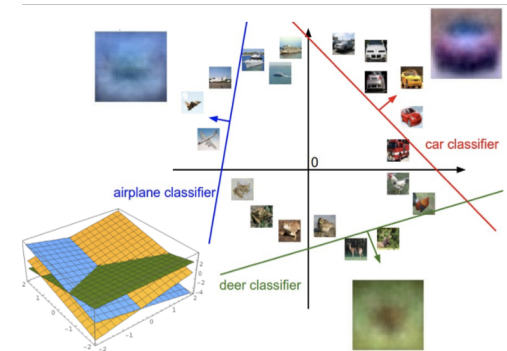
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



Recap: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function**:
- We have a **loss function**:

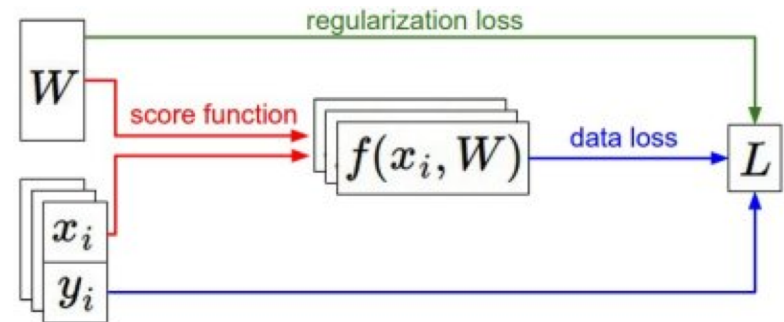
$$s = f(x; W) = Wx$$

Linear classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \text{ Softmax} \quad \text{SVM}$$

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

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



Recap: Loss Functions quantify preferences

- We have some dataset of (x, y)
- We have a **score function**:
- We have a **loss function**:

Q: How do we find the best W ?

$$s = f(x; W) = Wx$$

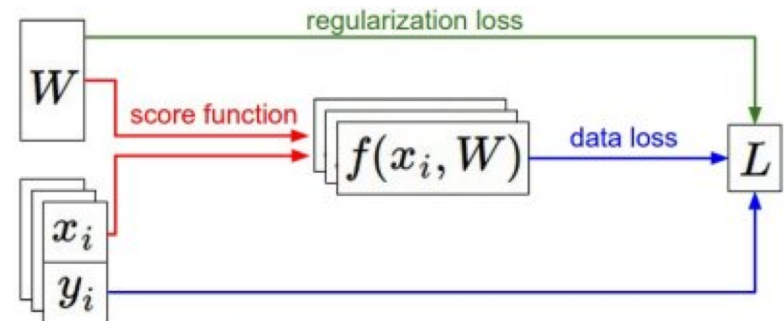
Linear classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \text{ Softmax}$$

SVM

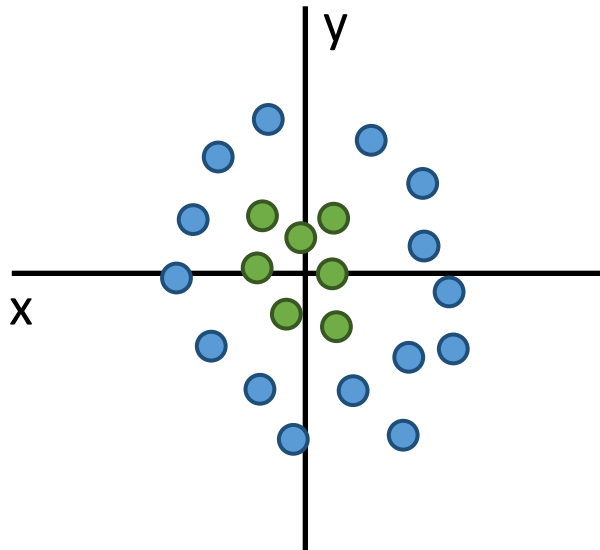
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



Problem: Linear Classifiers aren't that powerful

Geometric Viewpoint



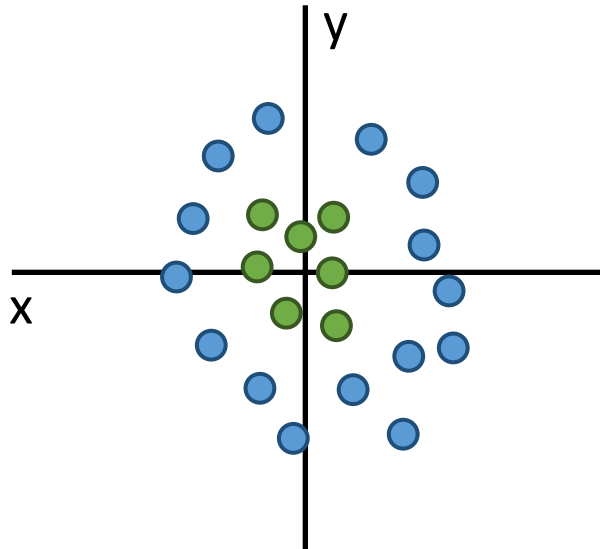
Visual Viewpoint

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



One solution: **Feature Transforms**

Original space

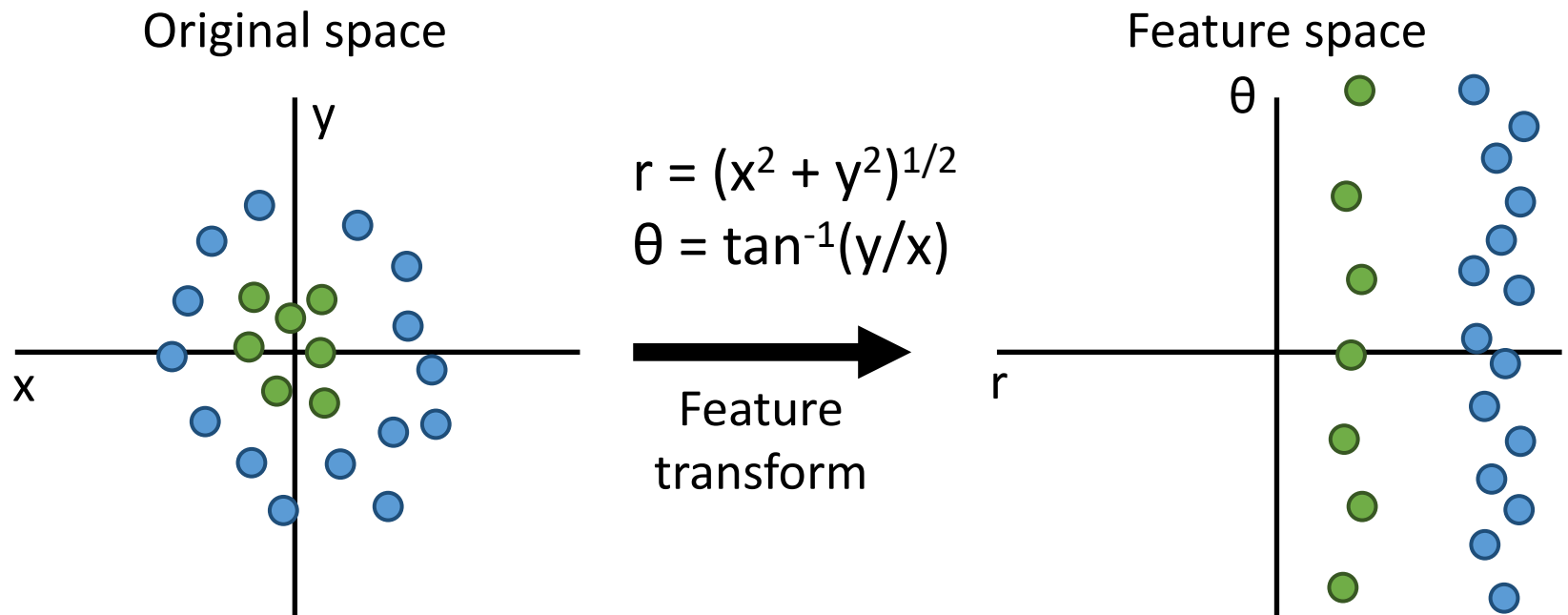


$$r = (x^2 + y^2)^{1/2}$$
$$\theta = \tan^{-1}(y/x)$$

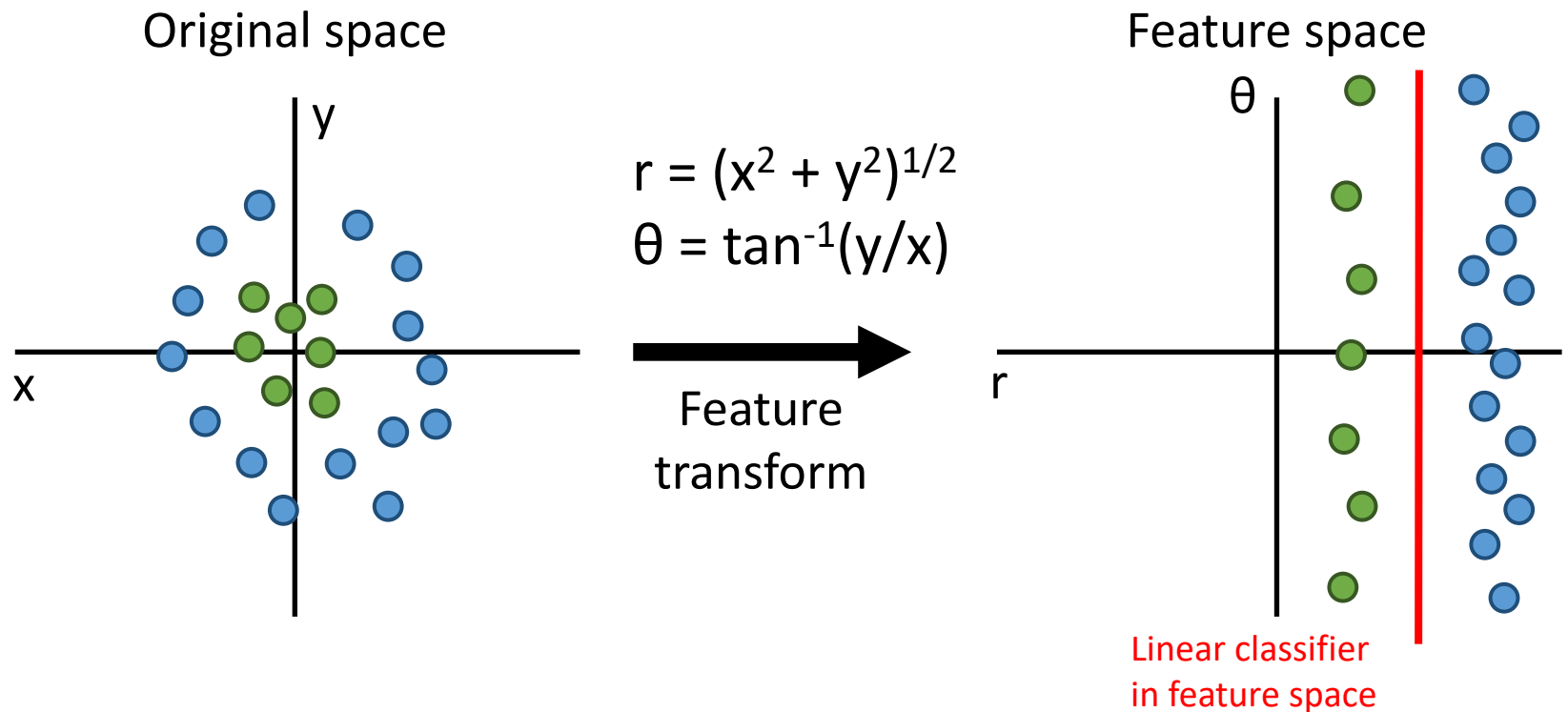


Feature
transform

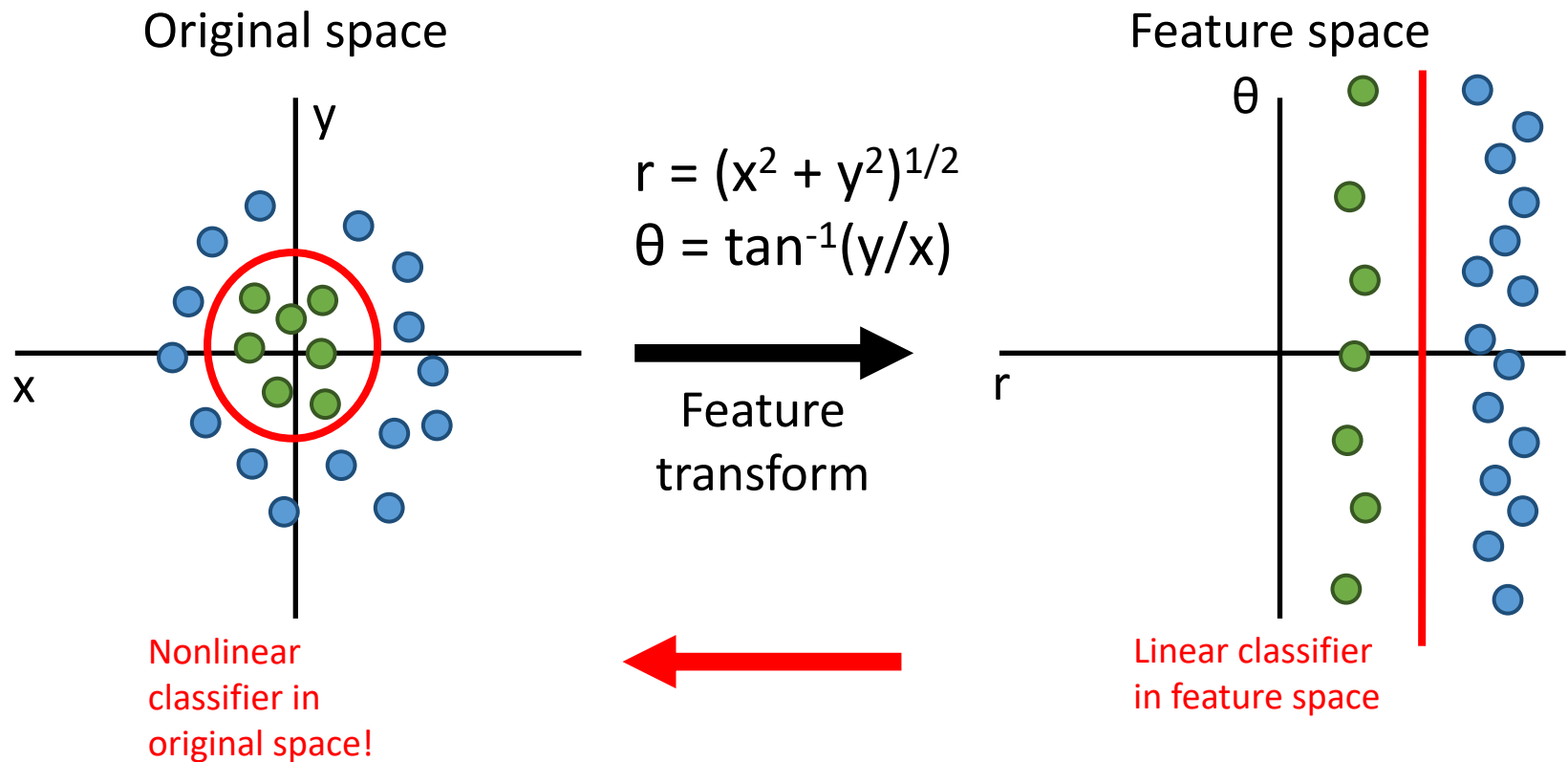
One solution: **Feature Transforms**



One solution: Feature Transforms

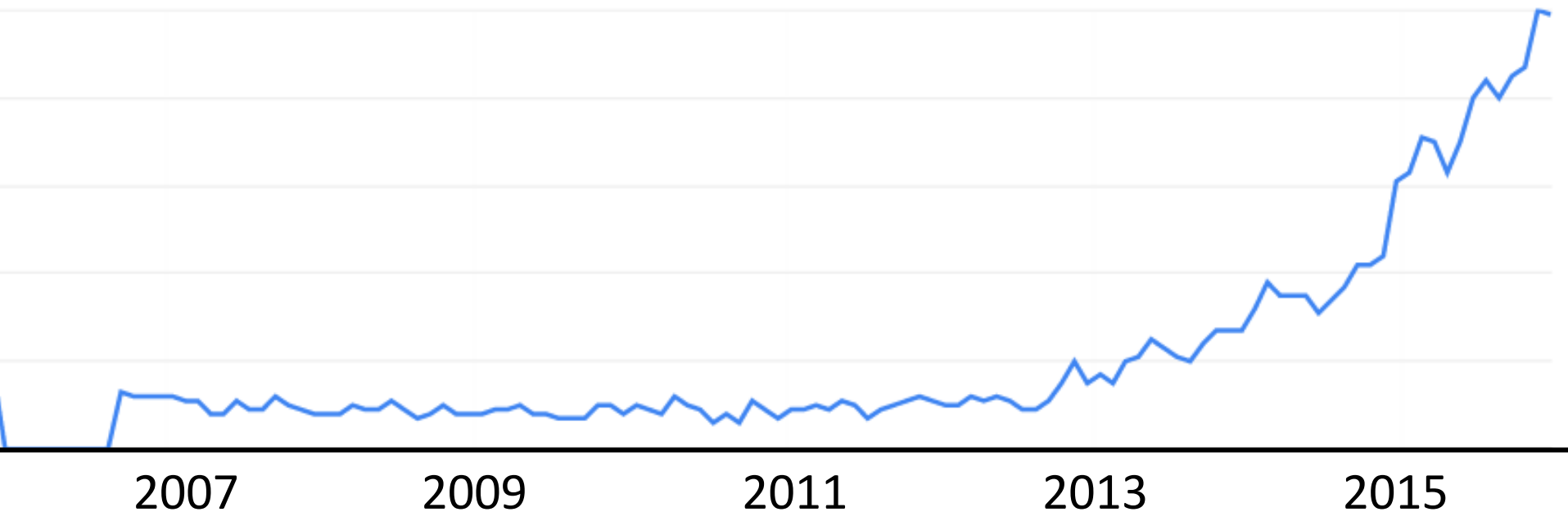


One solution: Feature Transforms

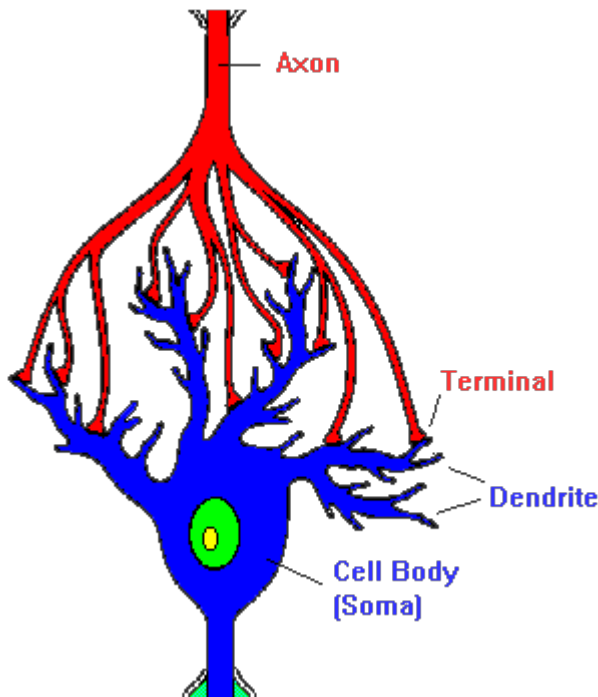
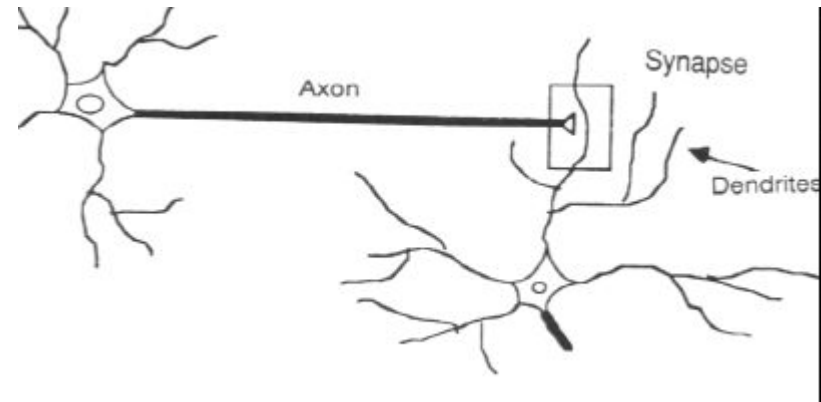
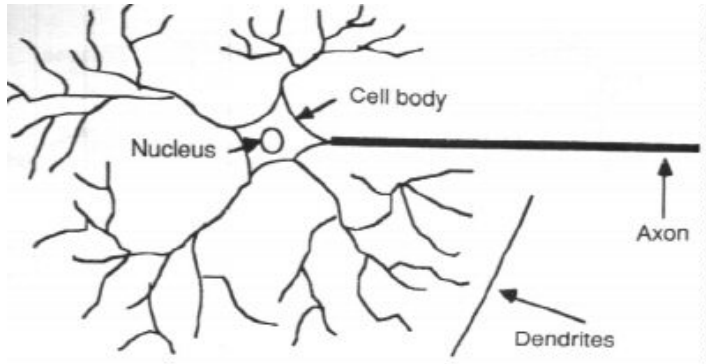


Deep learning attracts lots of attention.

- Google Trends

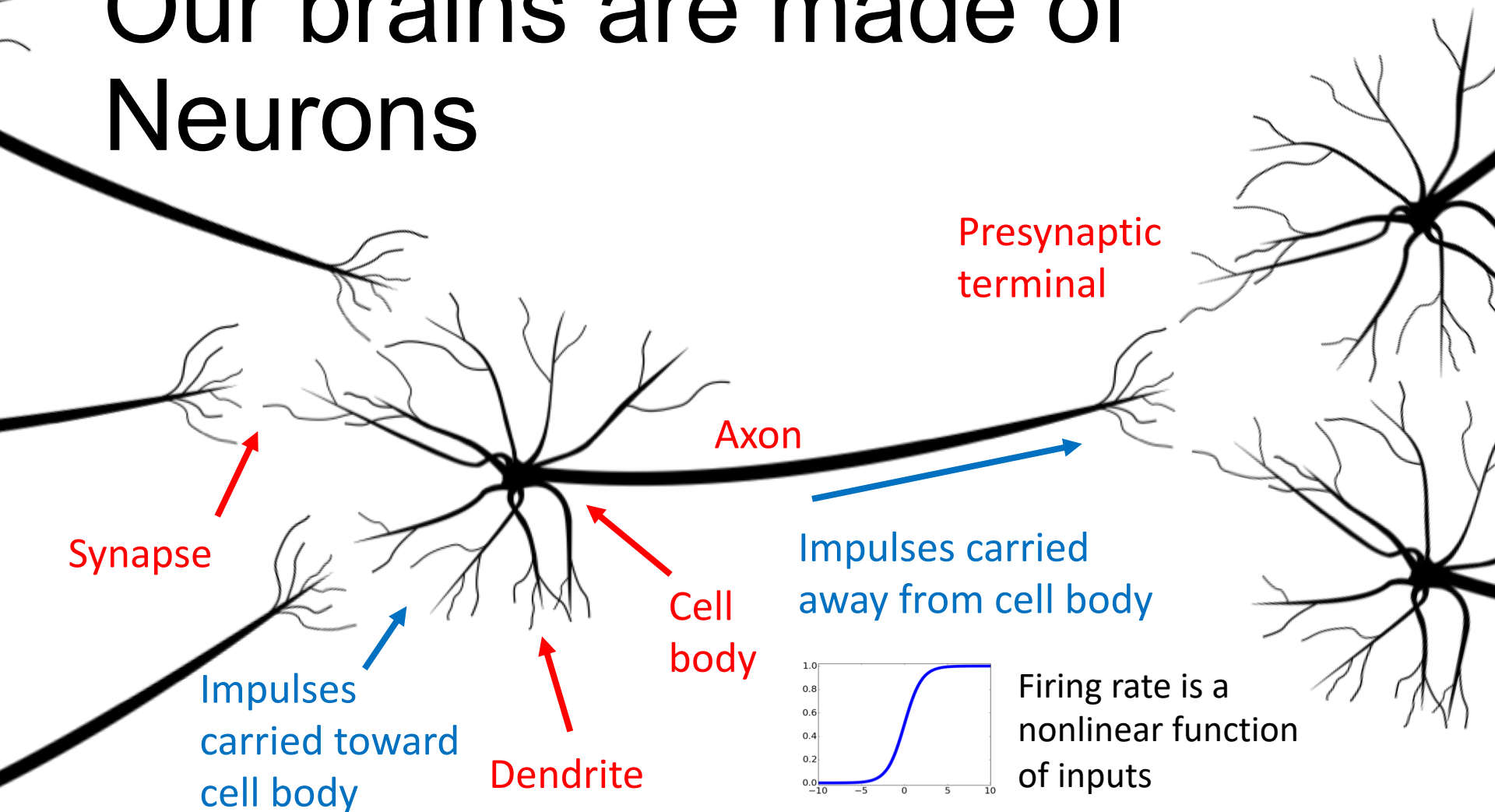


How the Human Brain learns

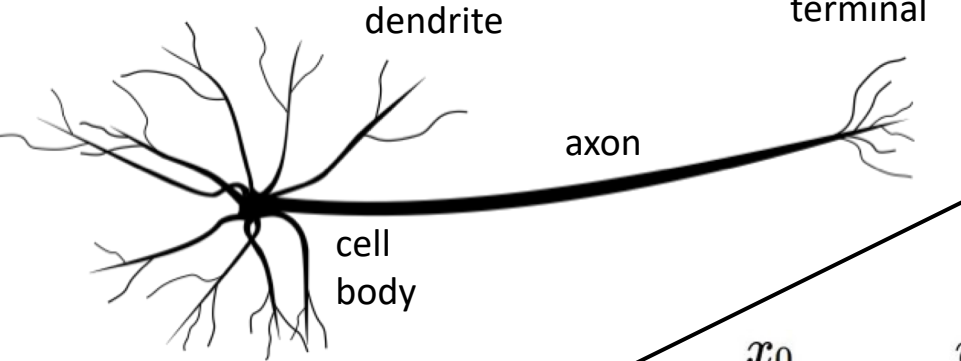


- In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.
- The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches.
- At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

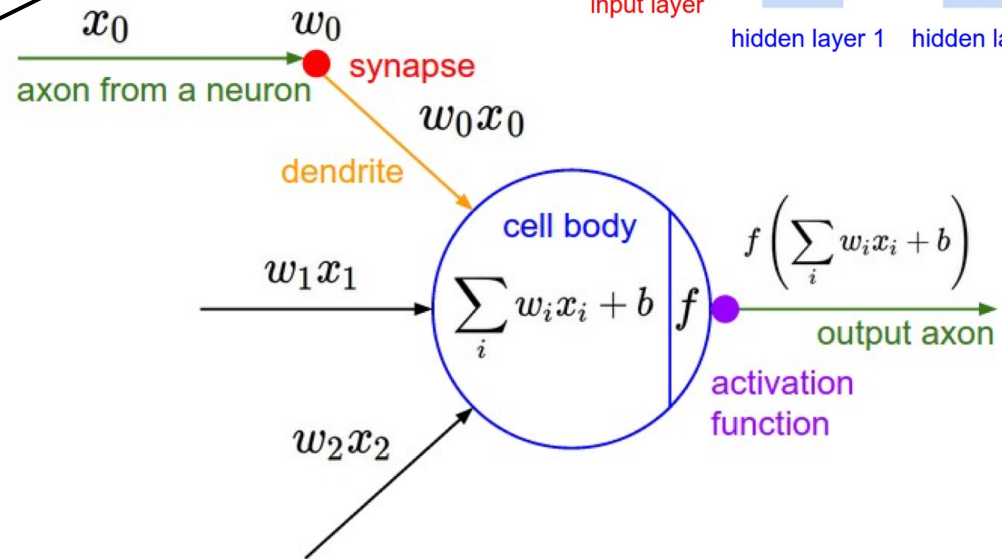
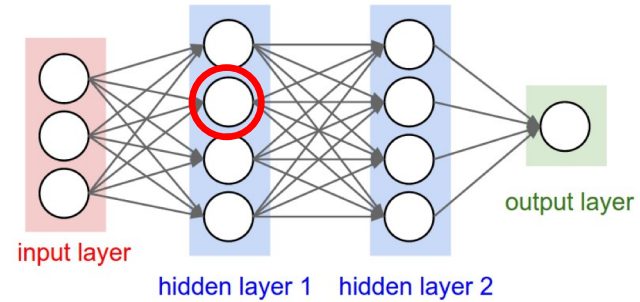
Our brains are made of Neurons



Biological Neuron



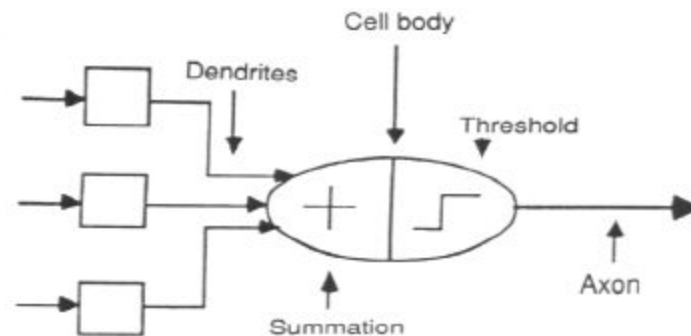
Artificial Neuron



Neuron image by Felipe Perucho is licensed under [CC-BY 3.0](https://creativecommons.org/licenses/by/3.0/)

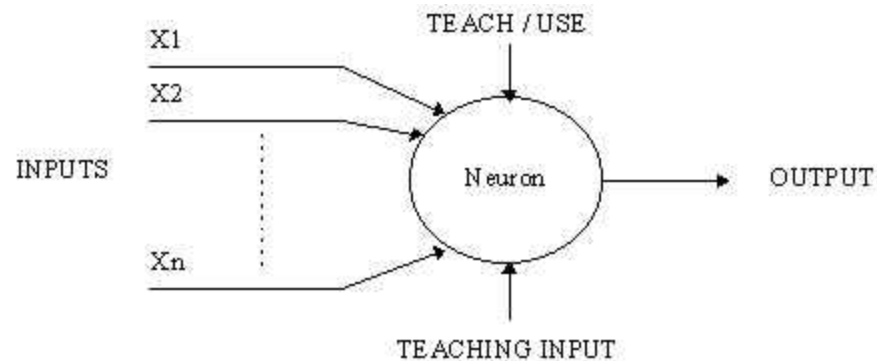
A Neuron Model

- When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.



- We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections.
- We then typically program a computer to simulate these features.

A Simple Neuron



- An artificial neuron is a device with many inputs and one output.
- The neuron has two modes of operation;
- the training mode and
- the using mode.

A Simple Neuron (Cont.)

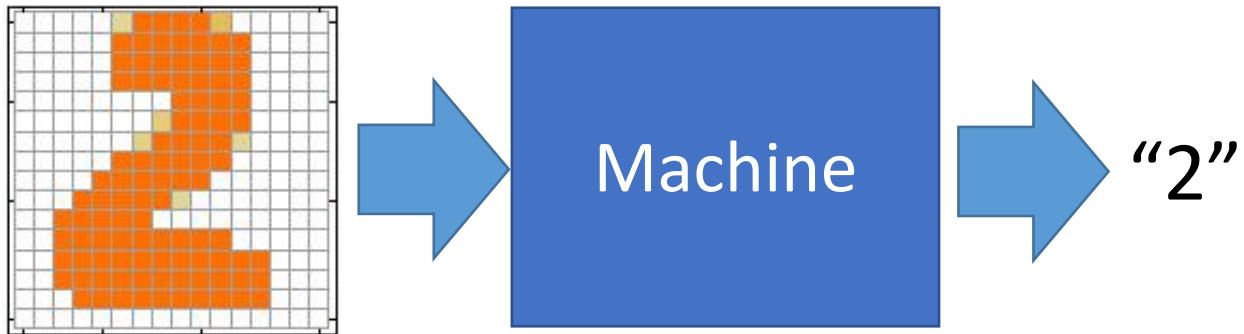
- In the training mode, the neuron can be trained to fire (or not), for particular input patterns.
- In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.
- The firing rule is an important concept in neural networks and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire for any input pattern. It relates to all the input patterns, not only the ones on which the node was trained on previously.

Part I: Introduction of Deep Learning

What people already knew in 1980s

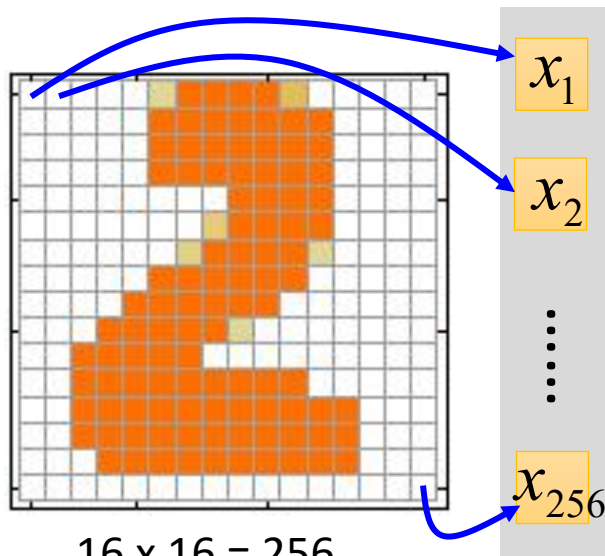
Example Application

- Handwriting Digit Recognition



Handwriting Digit Recognition

Input

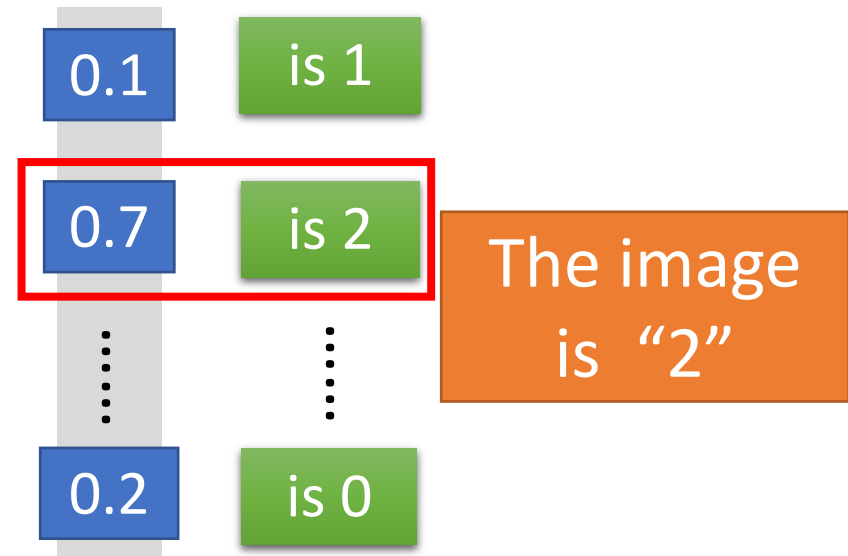


16 x 16 = 256

Ink \rightarrow 1

No ink \rightarrow 0

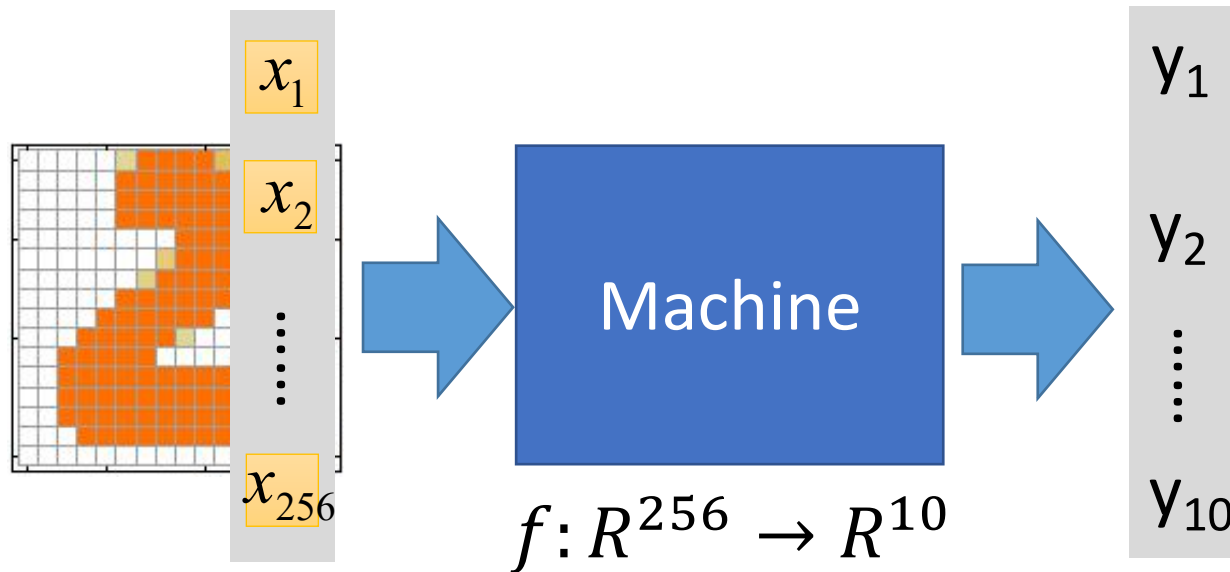
Output



Each dimension represents the confidence of a digit.

Example Application

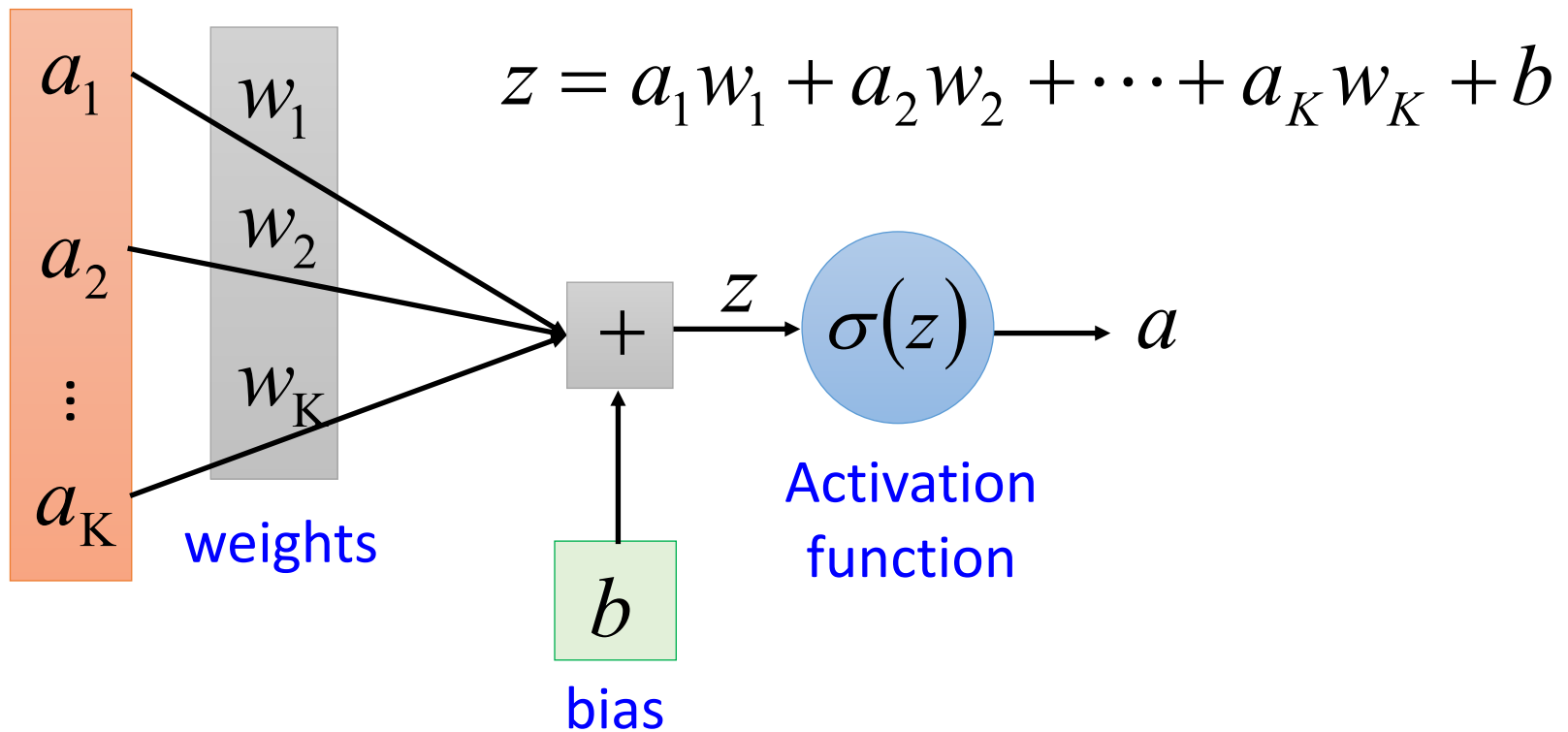
- Handwriting Digit Recognition



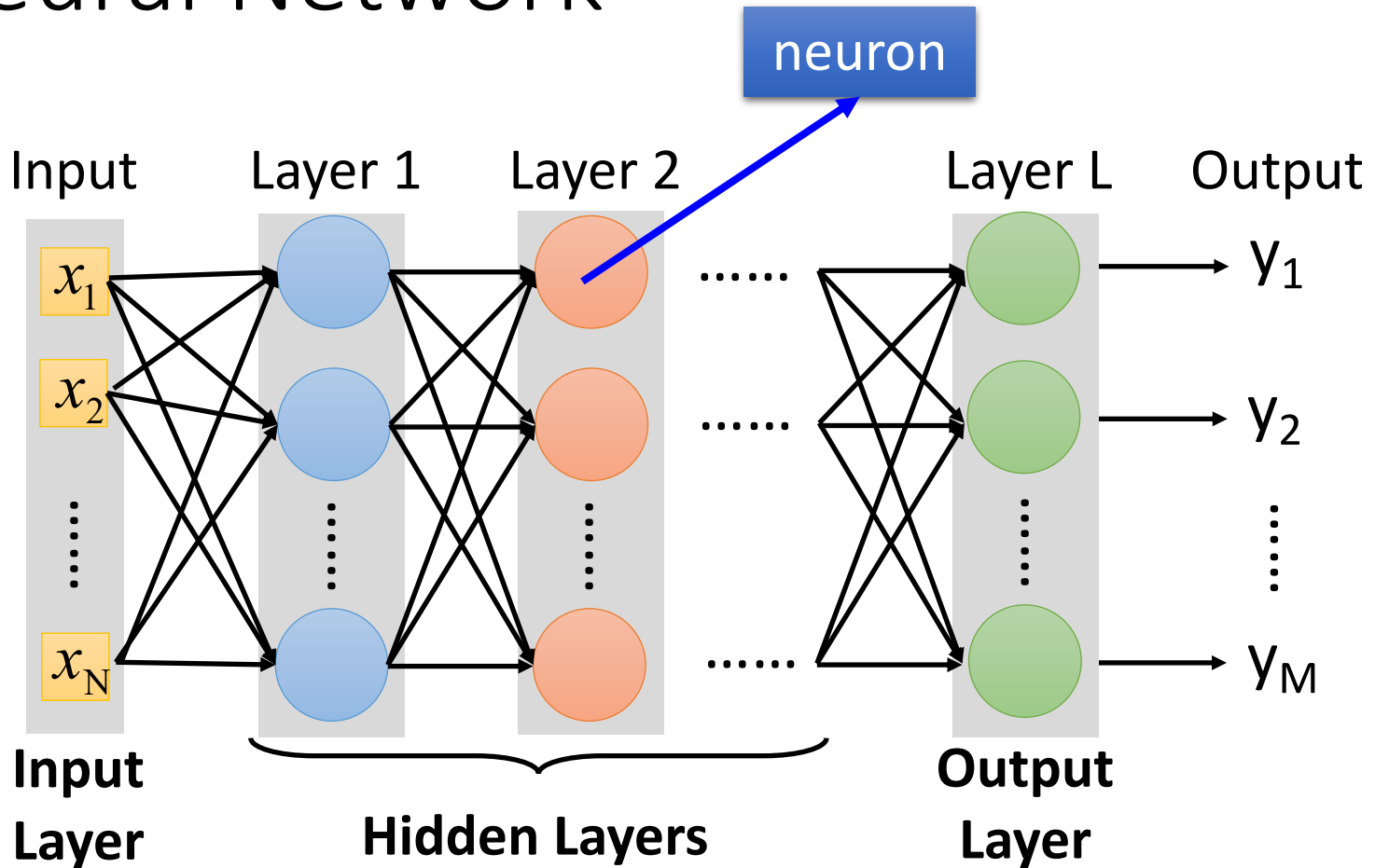
In deep learning, the function f is represented by neural network

Element of Neural Network

Neuron $f: R^K \rightarrow R$

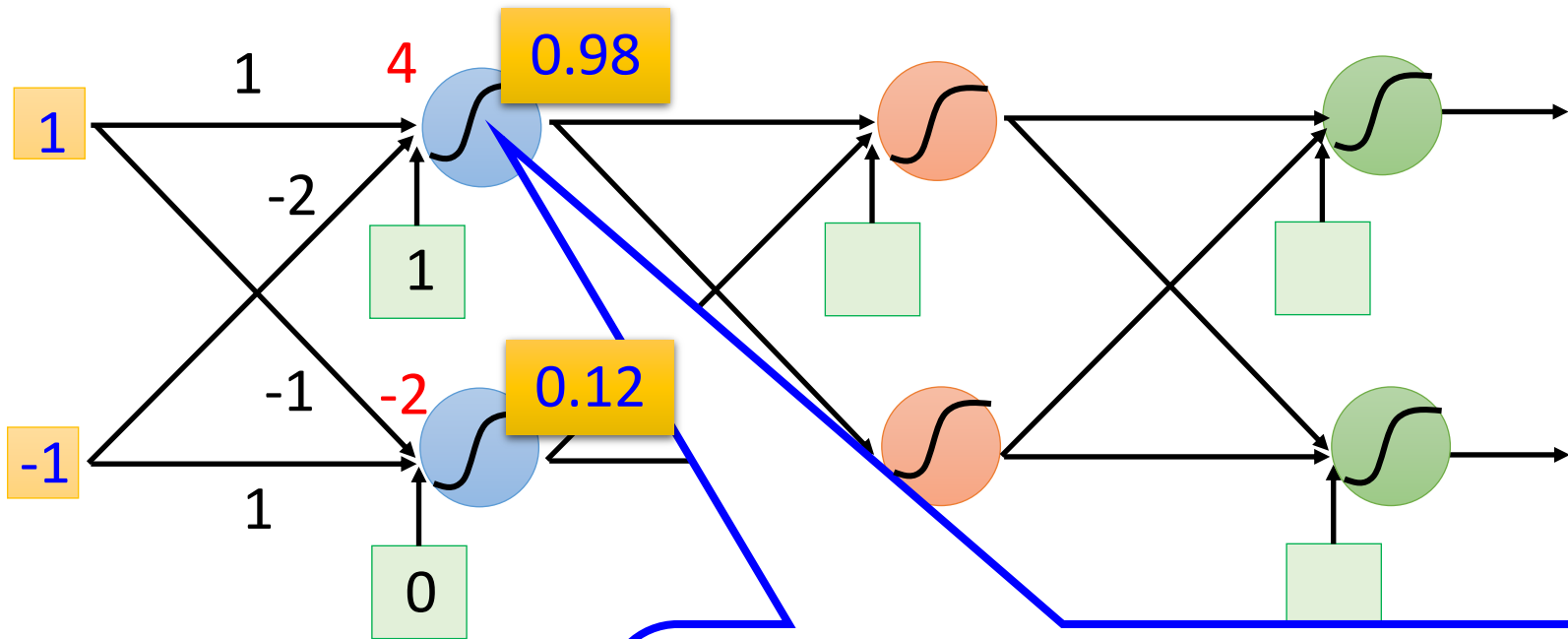


Neural Network



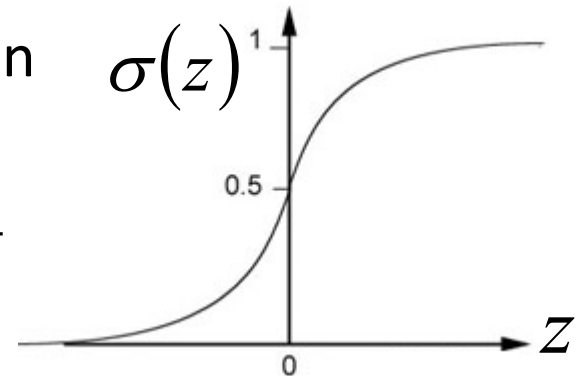
Deep means many hidden layers

Example of Neural Network

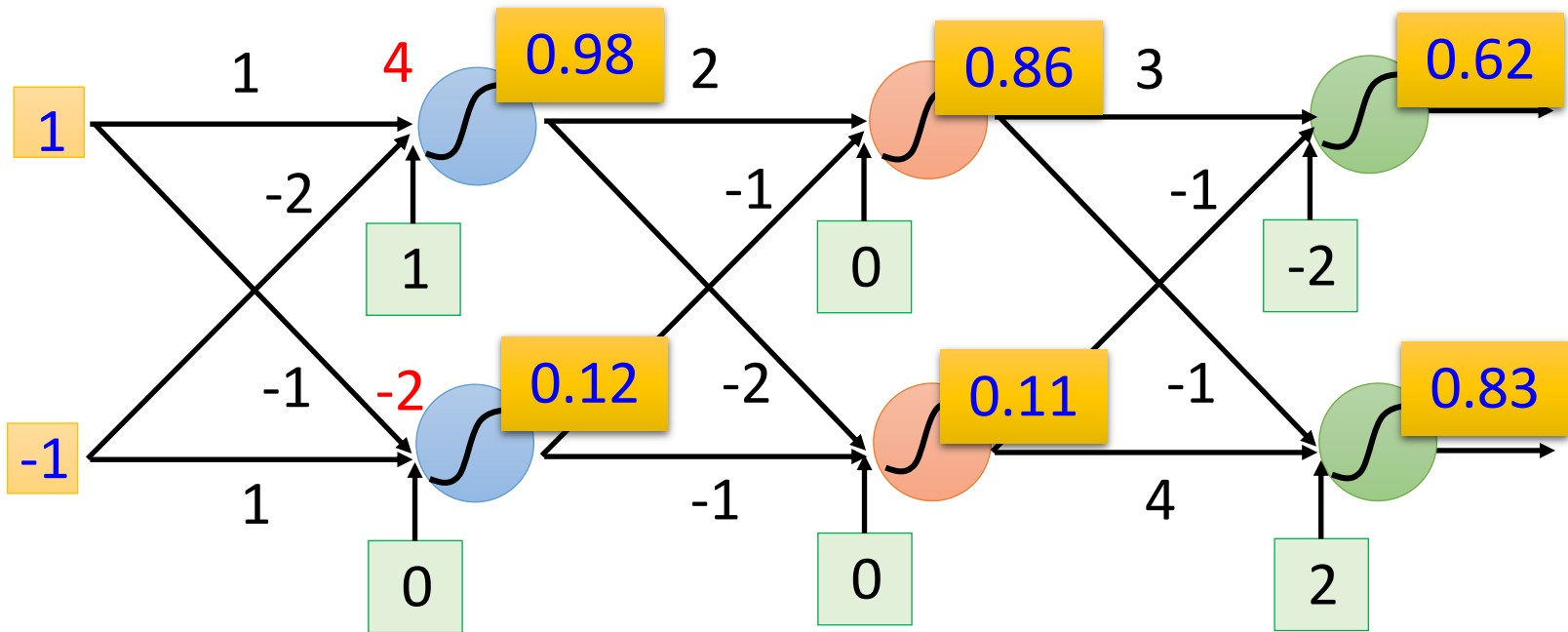


Sigmoid Function

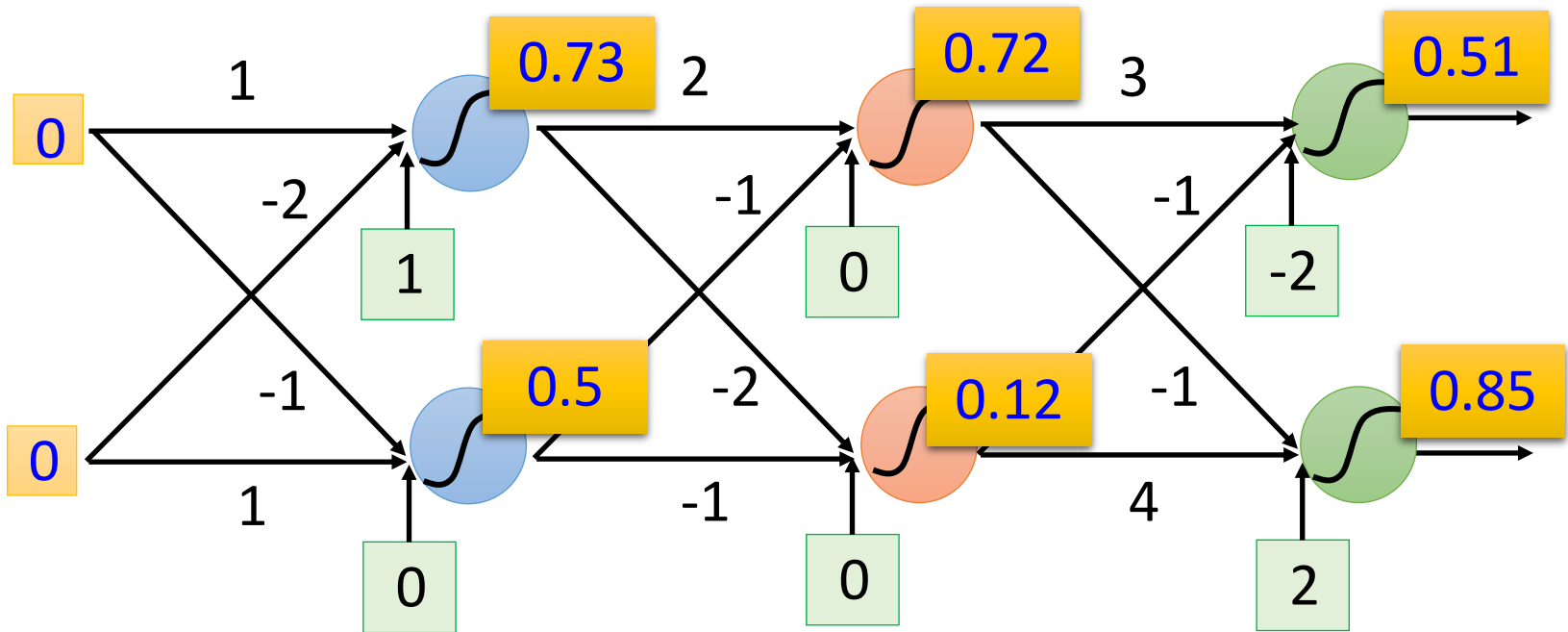
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Example of Neural Network



Example of Neural Network

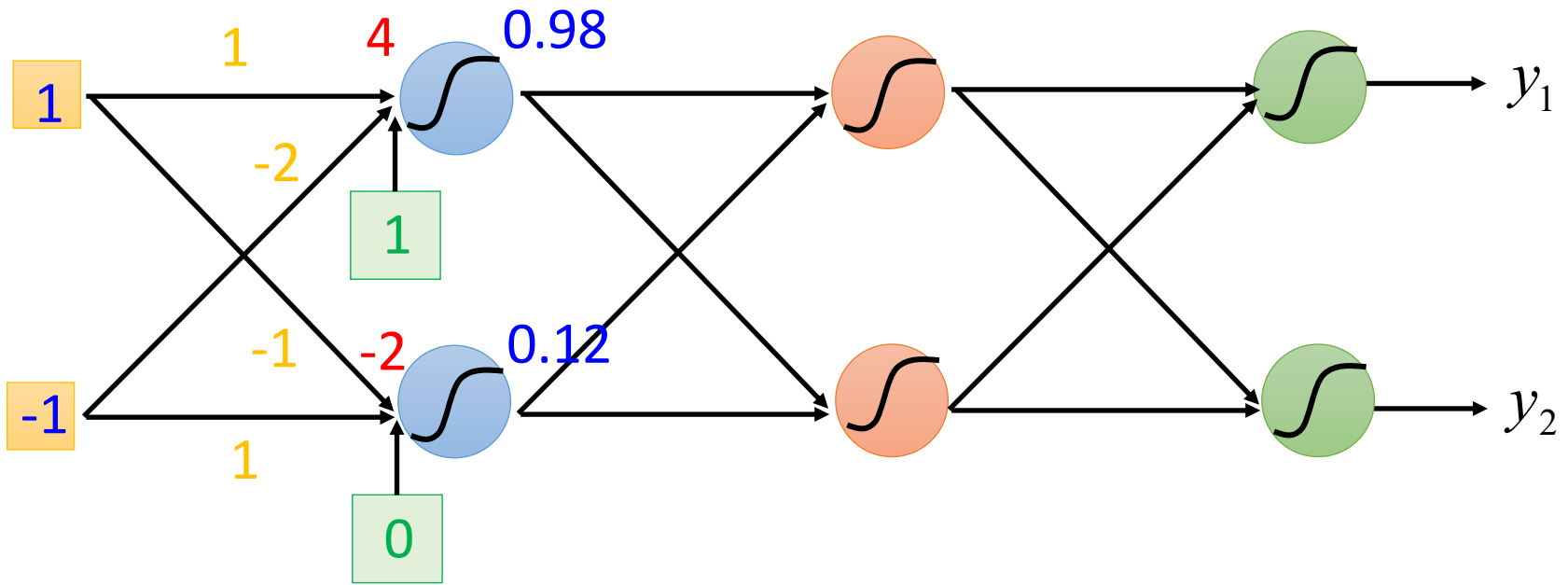


$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

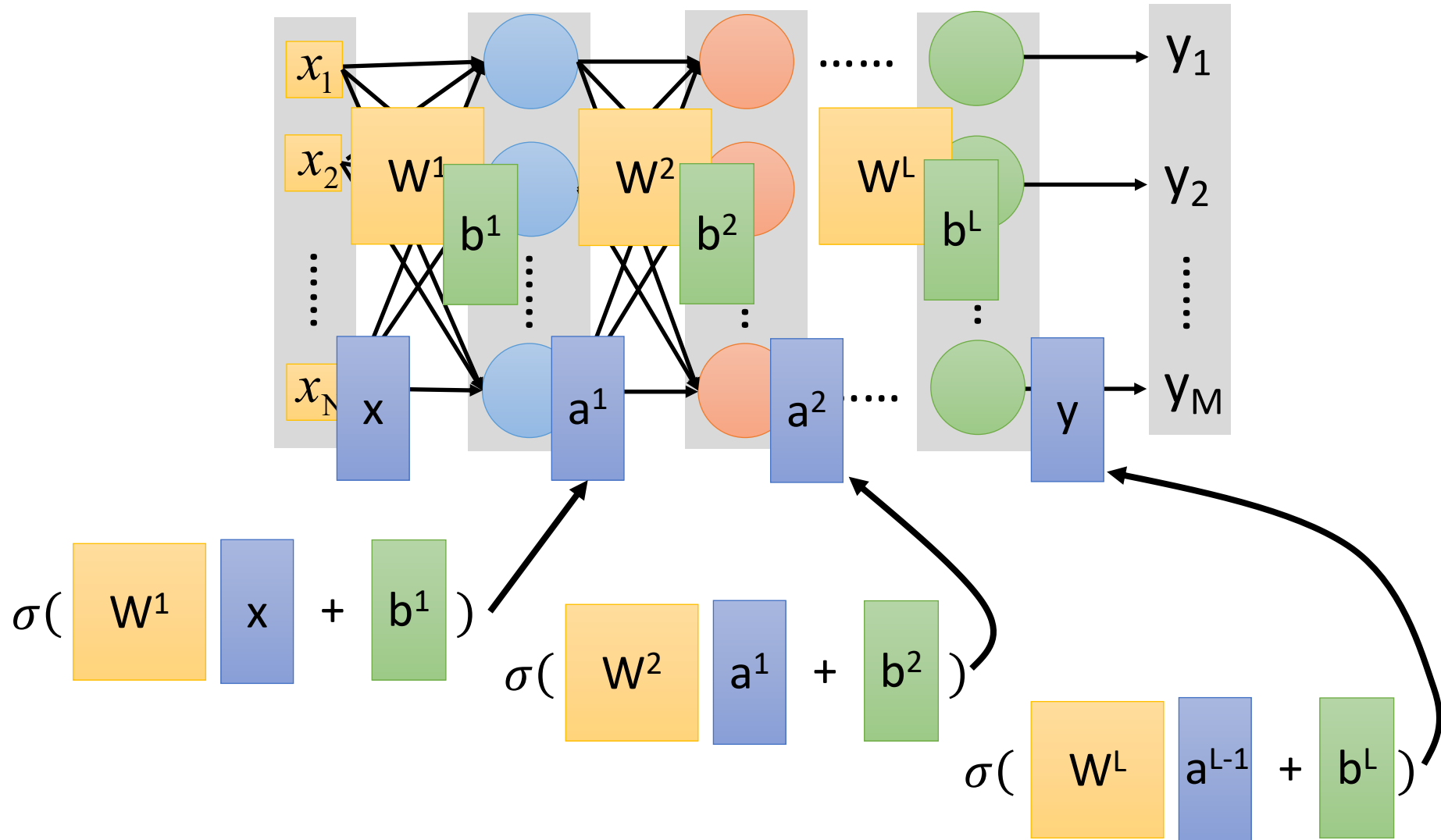
Different parameters define different function

Matrix Operation

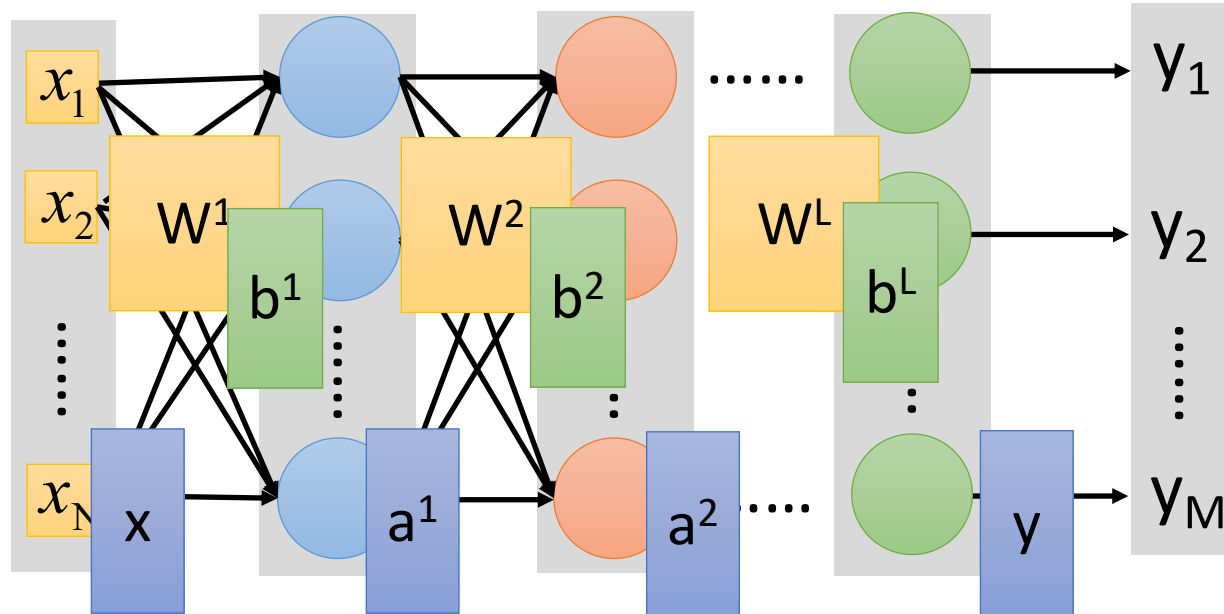


$$\sigma\left(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Neural Network



Neural Network



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

Softmax

- Softmax layer as the output layer

Ordinary Layer

$$z_1 \longrightarrow \sigma \longrightarrow y_1 = \sigma(z_1)$$

$$z_2 \longrightarrow \sigma \longrightarrow y_2 = \sigma(z_2)$$

$$z_3 \longrightarrow \sigma \longrightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

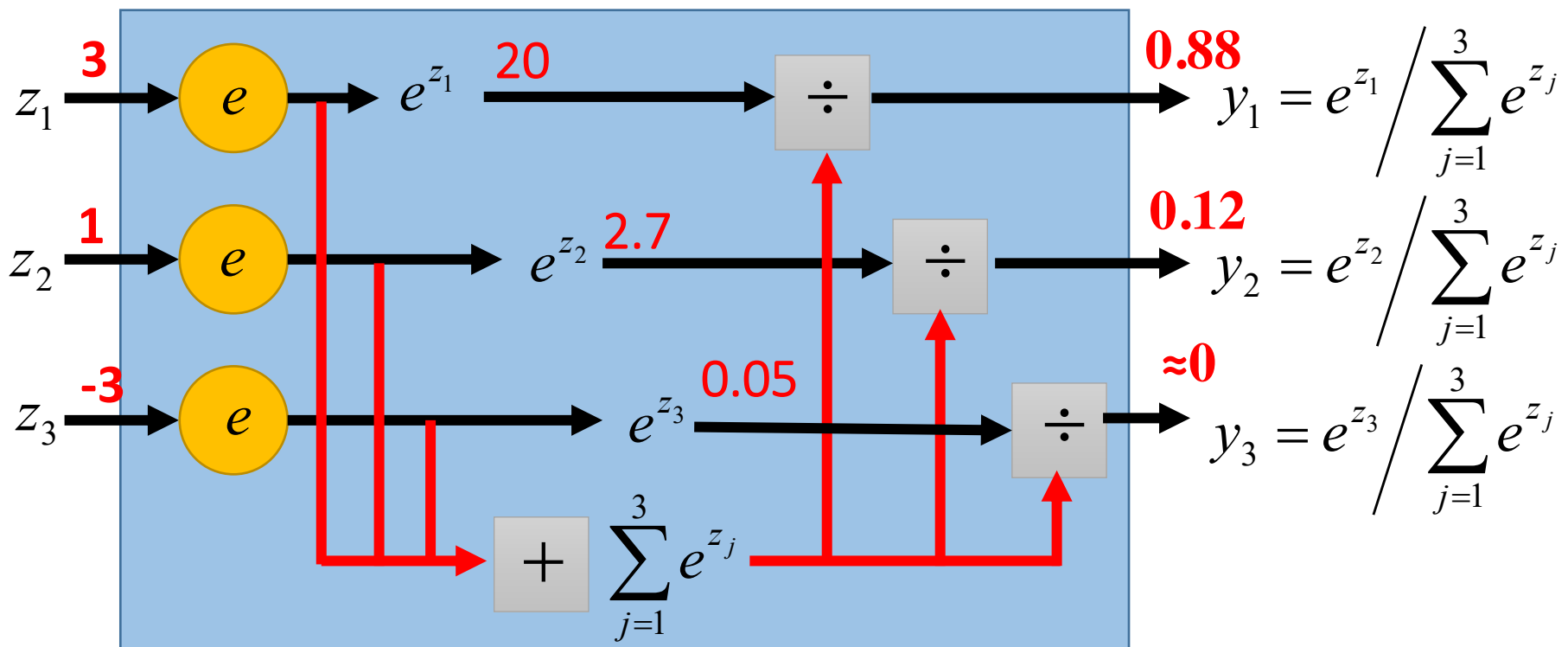
Softmax

- Softmax layer as the output layer

Probability:

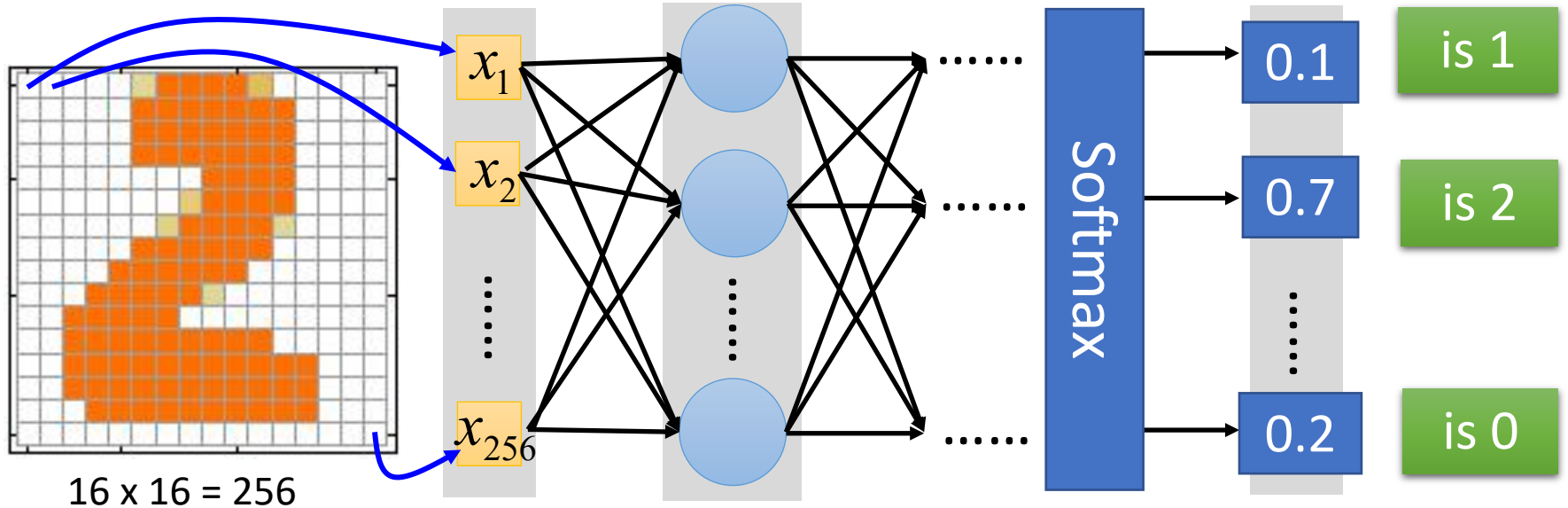
- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$



Set the network parameters θ such that

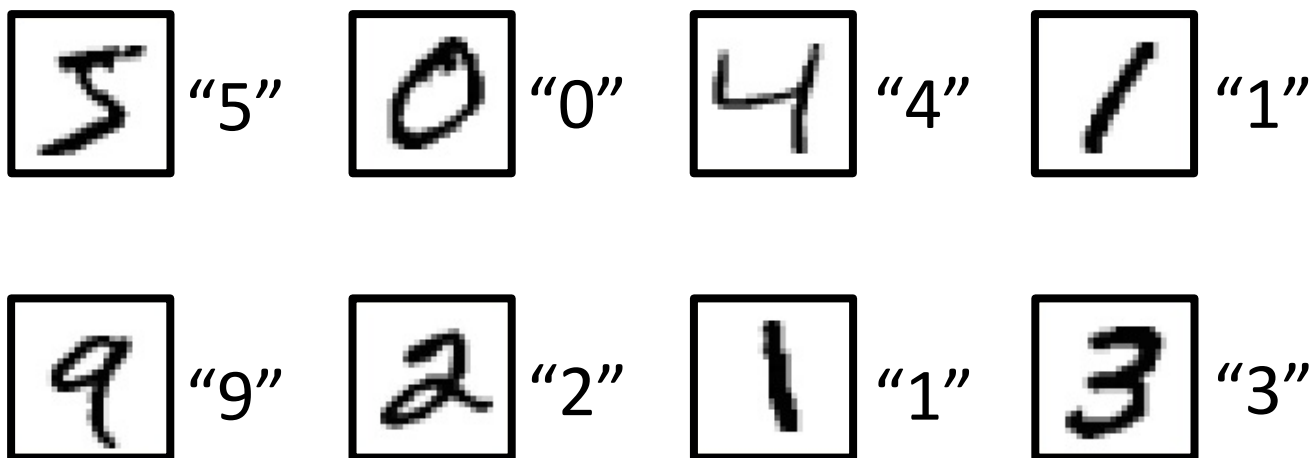
Input:  y_2 has the maximum value

Input:  y_2 has the maximum value

How to let the neural network achieve this

Training Data

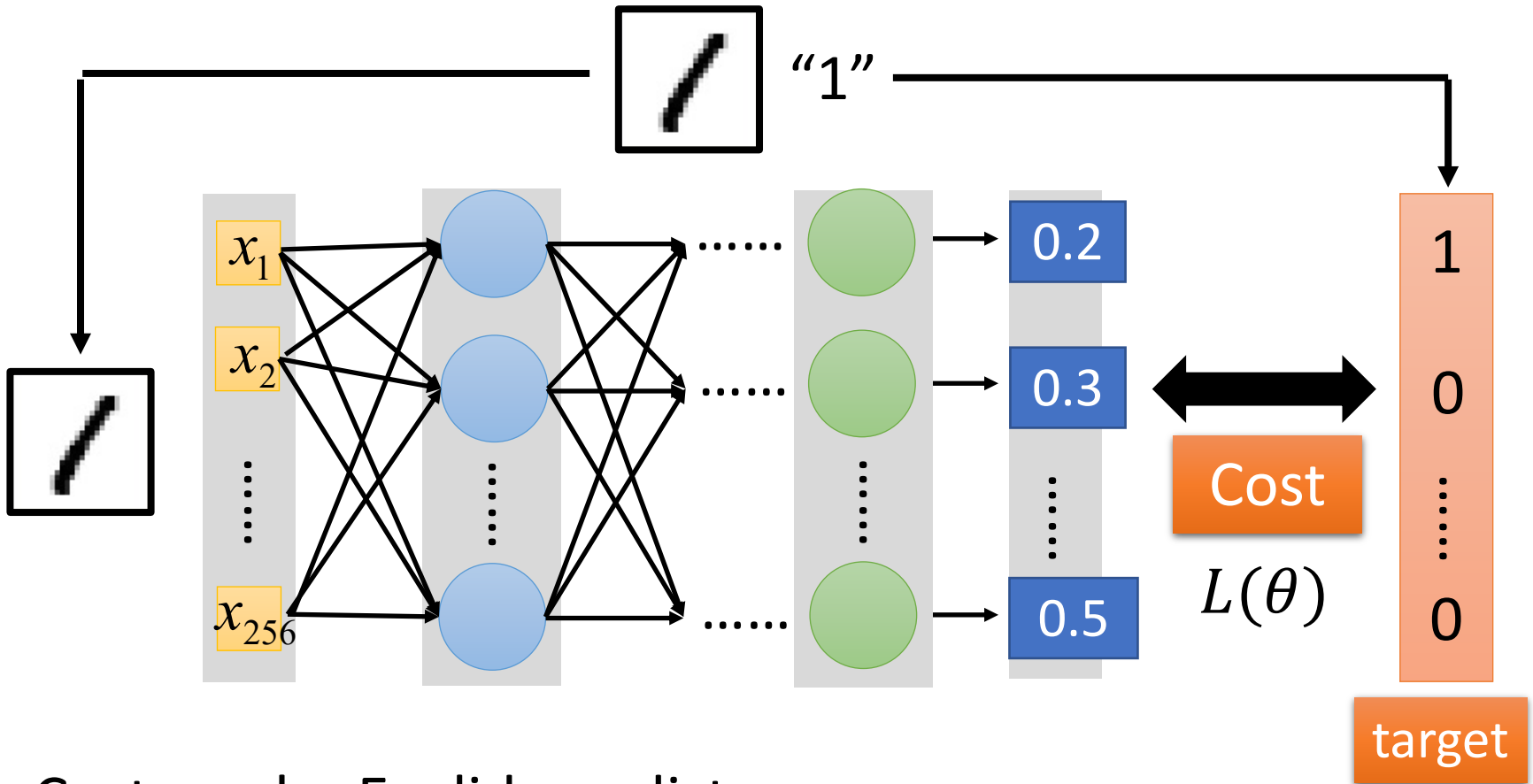
- Preparing training data: images and their labels



Using the training data to find
the network parameters.

Cost

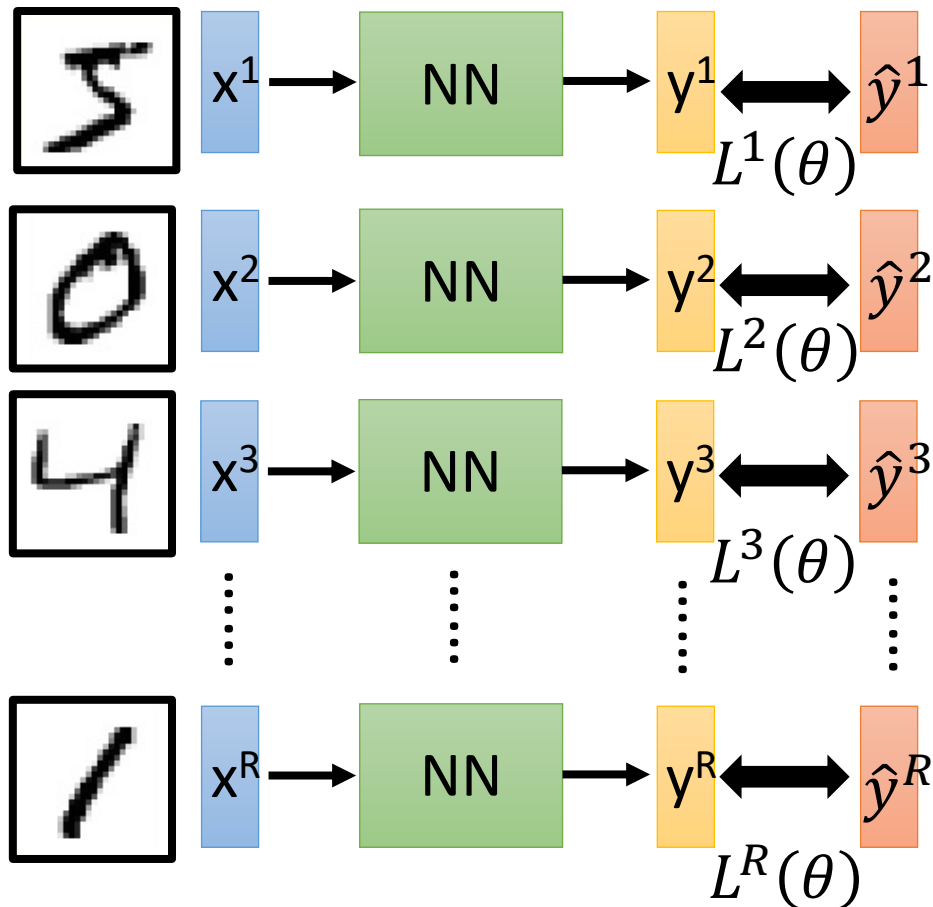
Given a set of network parameters θ , each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data ...



Total Cost:

$$C(\theta) = \sum_{r=1}^R L^r(\theta)$$

How bad the network parameters θ is on this task

Find the network parameters θ^* that minimize this value

Gradient Descent


Error Surface

Assume there are only two parameters w_1 and w_2 in a network.


$$\theta = \{w_1, w_2\}$$

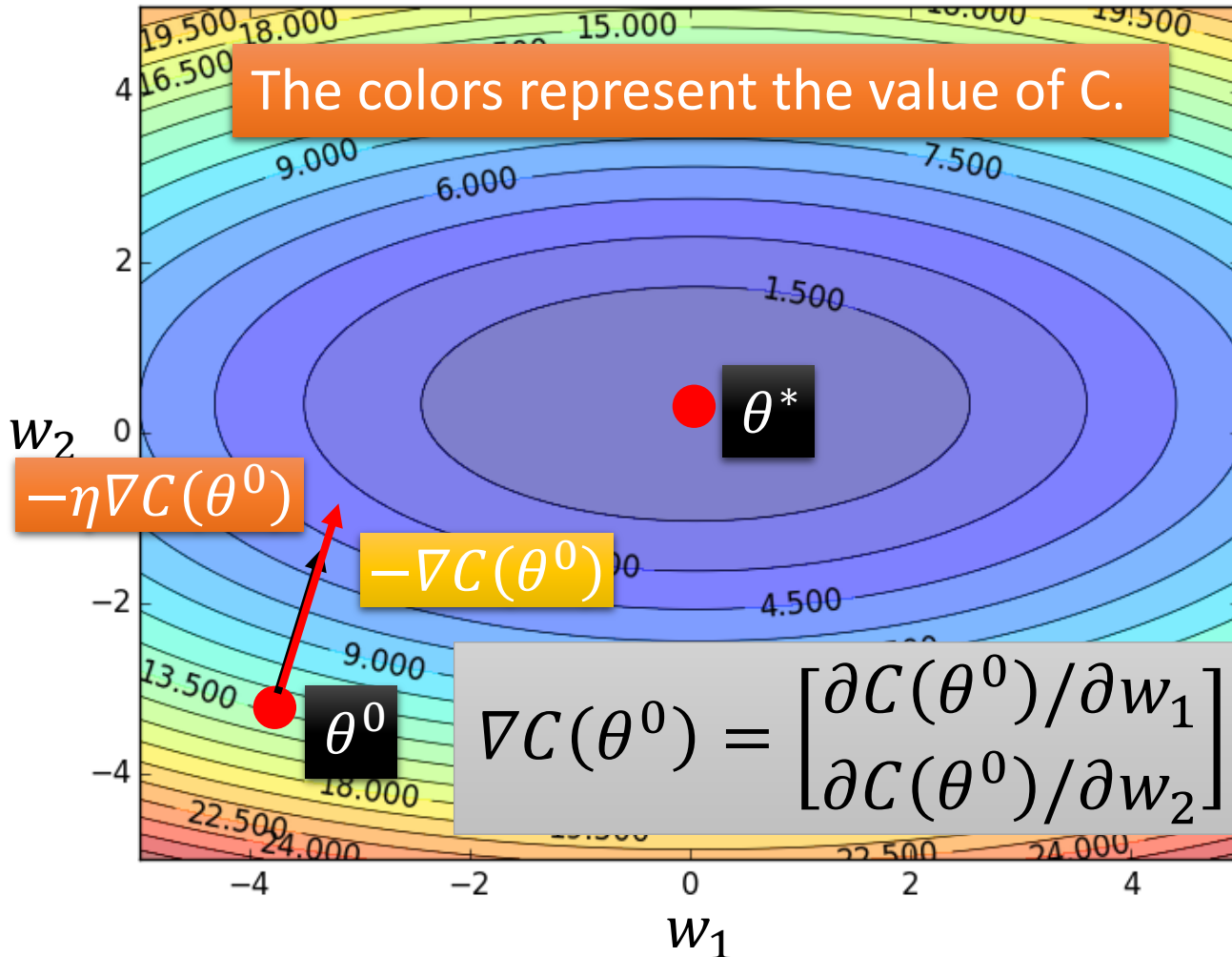
Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

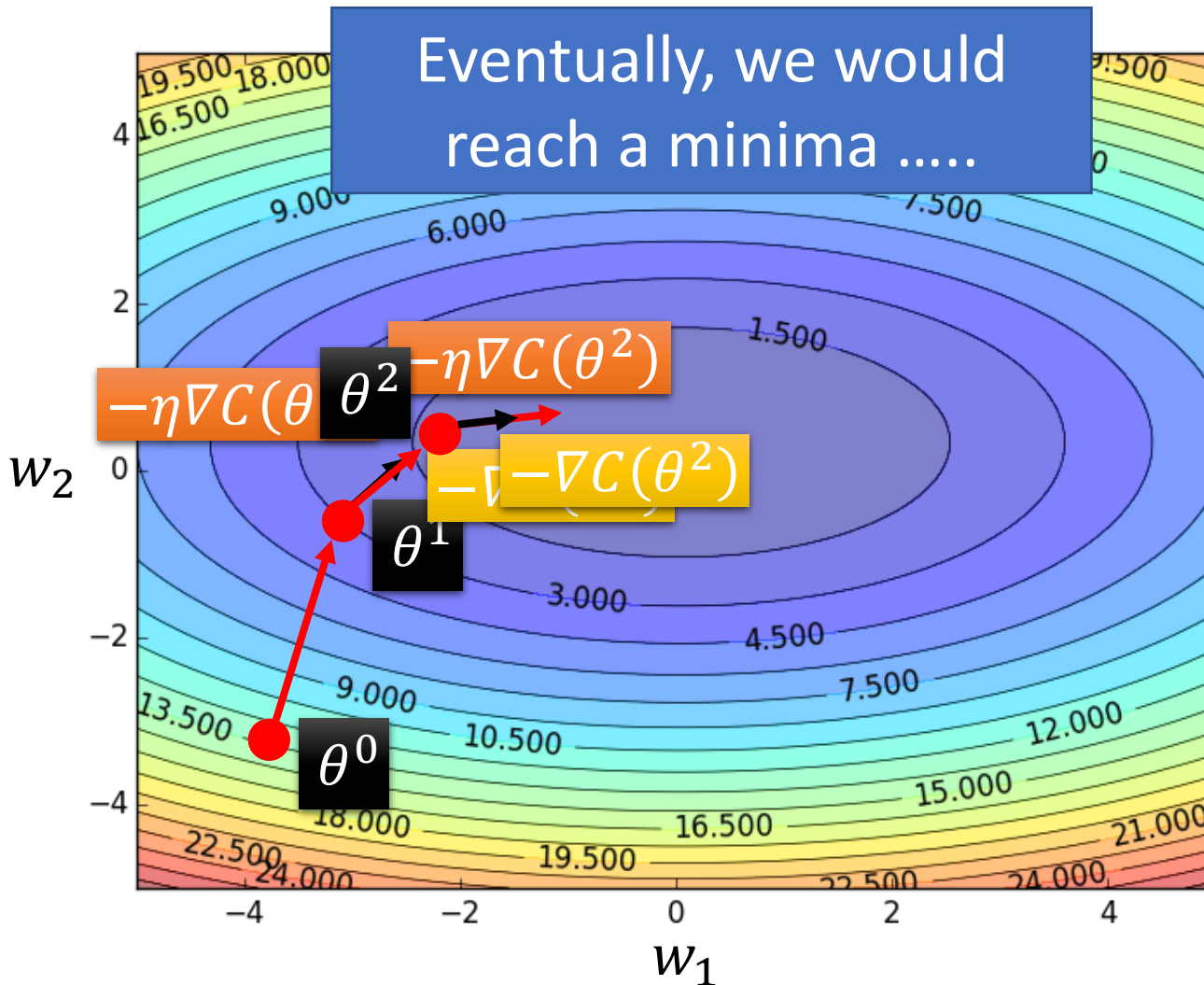
 $-\nabla C(\theta^0)$

Times the learning rate η

 $-\eta \nabla C(\theta^0)$



Gradient Descent



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

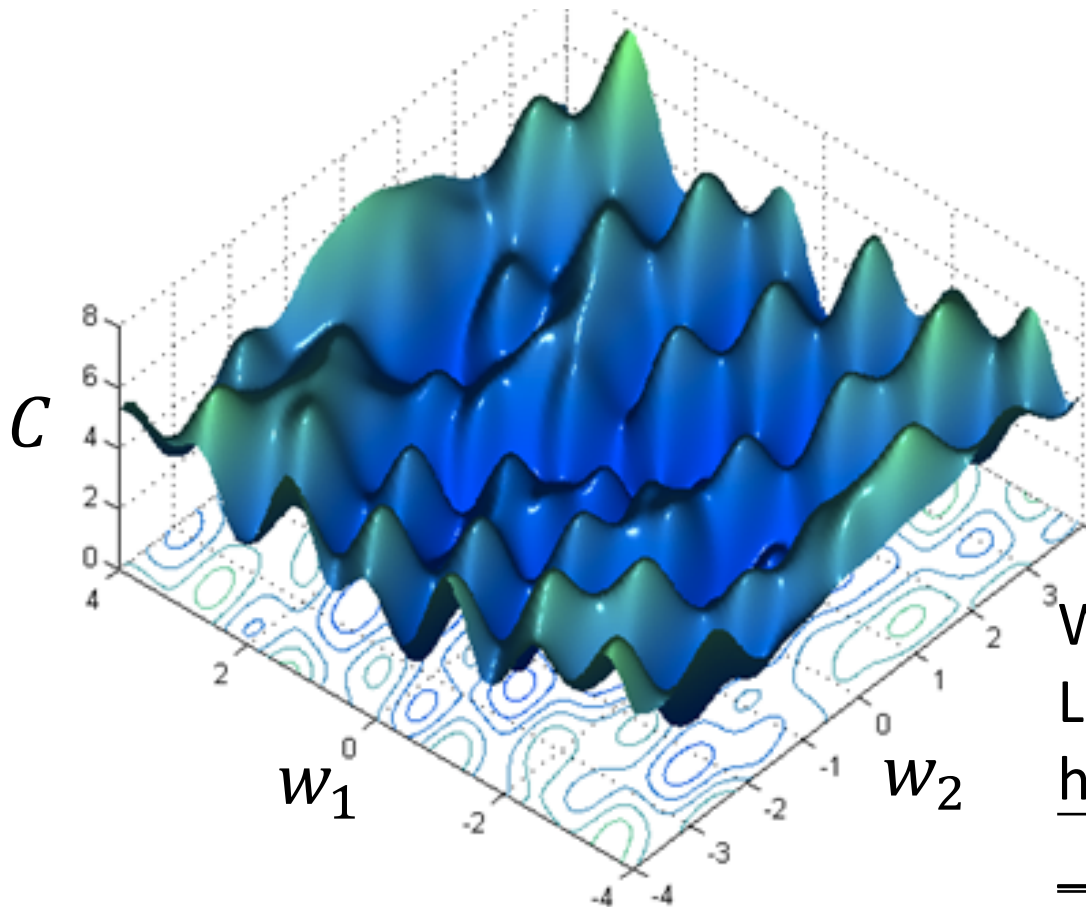
$\rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\rightarrow -\eta \nabla C(\theta^0)$

Local Minima

- Gradient descent never guarantee global minima



Different initial
point θ^0

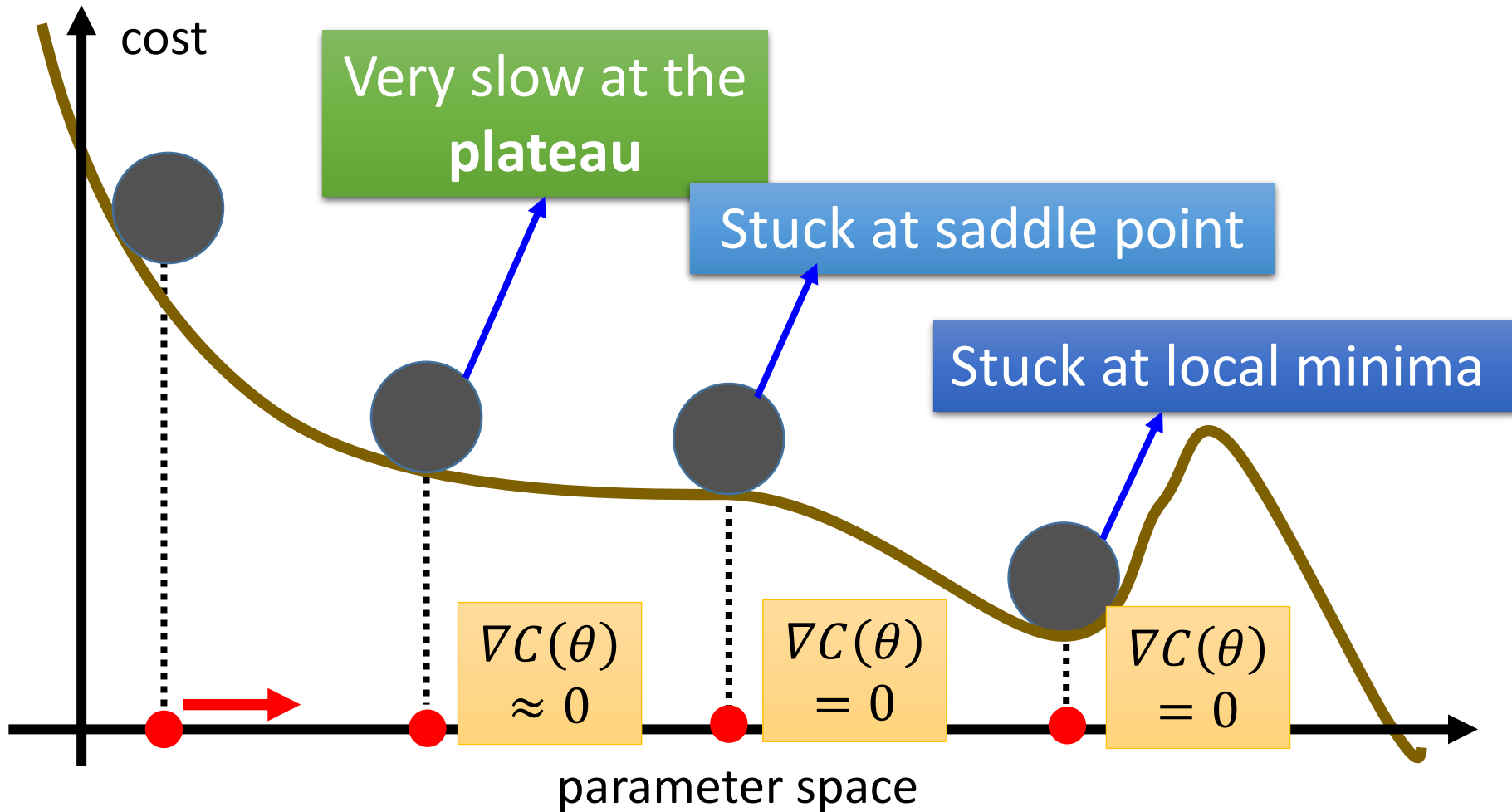


Reach different minima,
so different results

Who is Afraid of Non-Convex
Loss Functions?

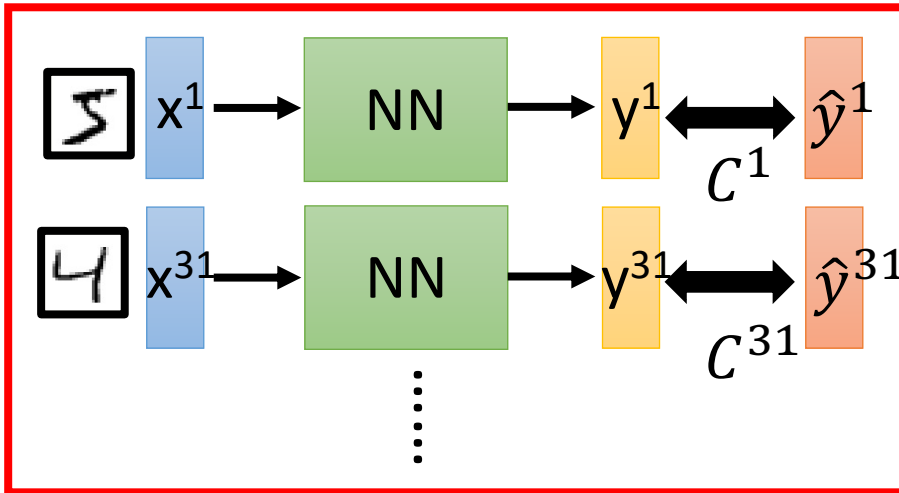
http://videolectures.net/eml07_lecun_wia/

Besides local minima

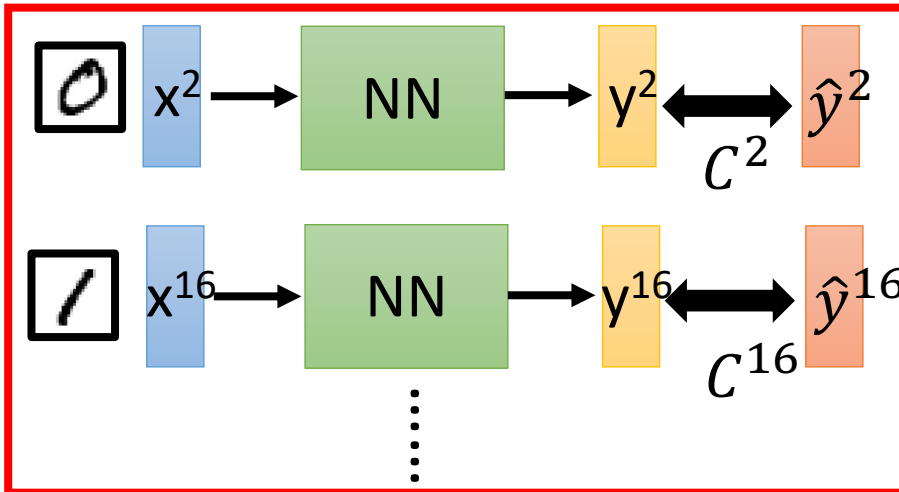


Mini-batch

Mini-batch



Mini-batch



➤ Randomly initialize θ^0

➤ Pick the 1st batch

$$C = C^1 + C^{31} + \dots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

➤ Pick the 2nd batch

$$C = C^2 + C^{16} + \dots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

⋮

➤ Until all mini-batches have been picked

one epoch

Repeat the above process

Backpropagation

- A network can have millions of parameters.
 - Backpropagation is the way to compute the gradients efficiently (not today)
 - Ref:
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html
- Many toolkits can compute the gradients automatically

theano



Ref:

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

Size of Training Data

- Rule of thumb:

- the number of training examples should be at least five to ten times the number of weights of the network.

- Other rule:

$$N > \frac{|W|}{(1 - a)}$$

$|W|$ = number of weights

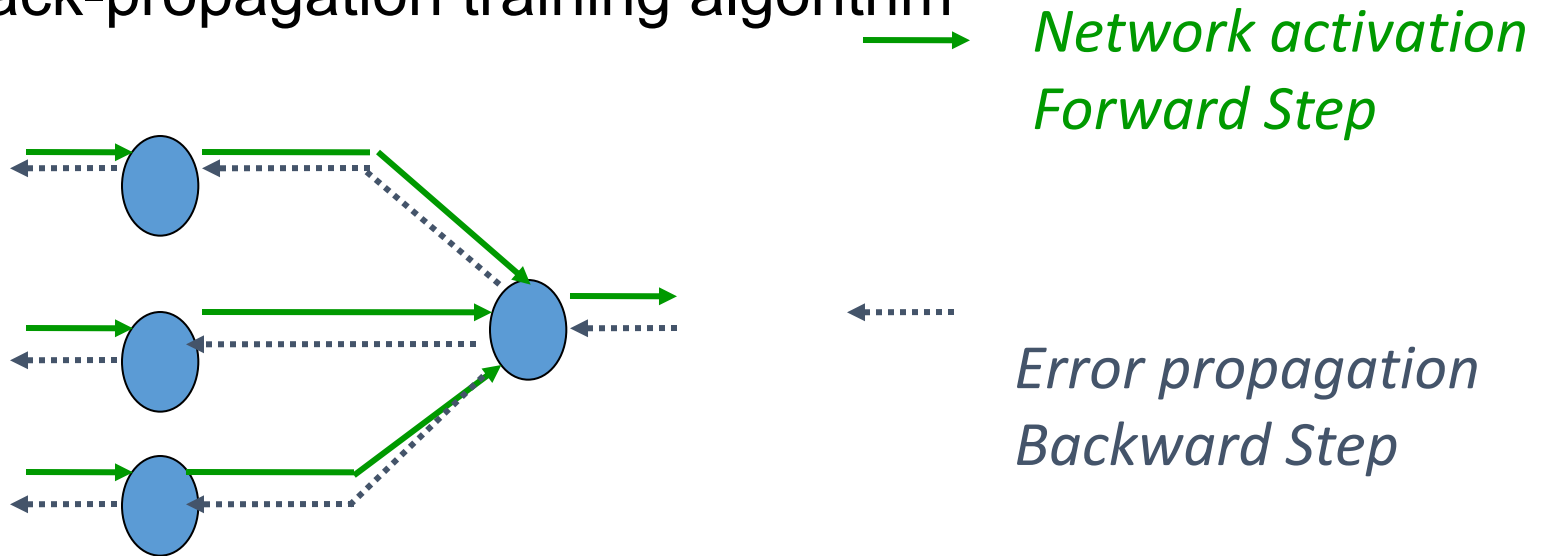
a = expected accuracy on test set

Training: Backprop algorithm

- The Backprop algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backprop consists of the repeated application of the following two passes:
 - **Forward pass:** in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
 - **Backward pass:** in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.

Back Propagation

- Back-propagation training algorithm



- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

Part II:
Why Deep?

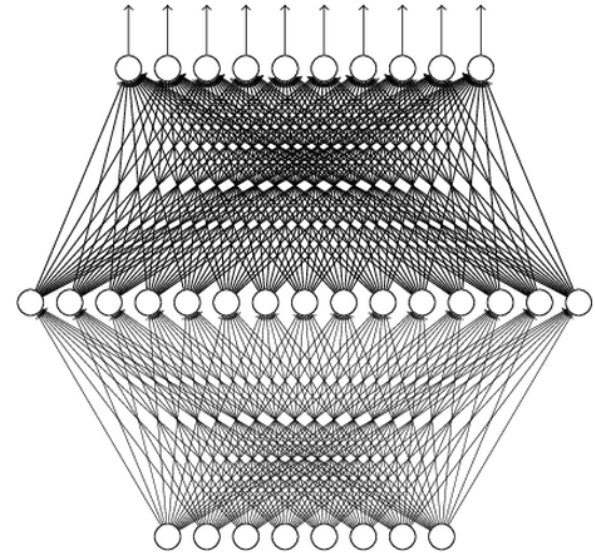
Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network
with one hidden layer

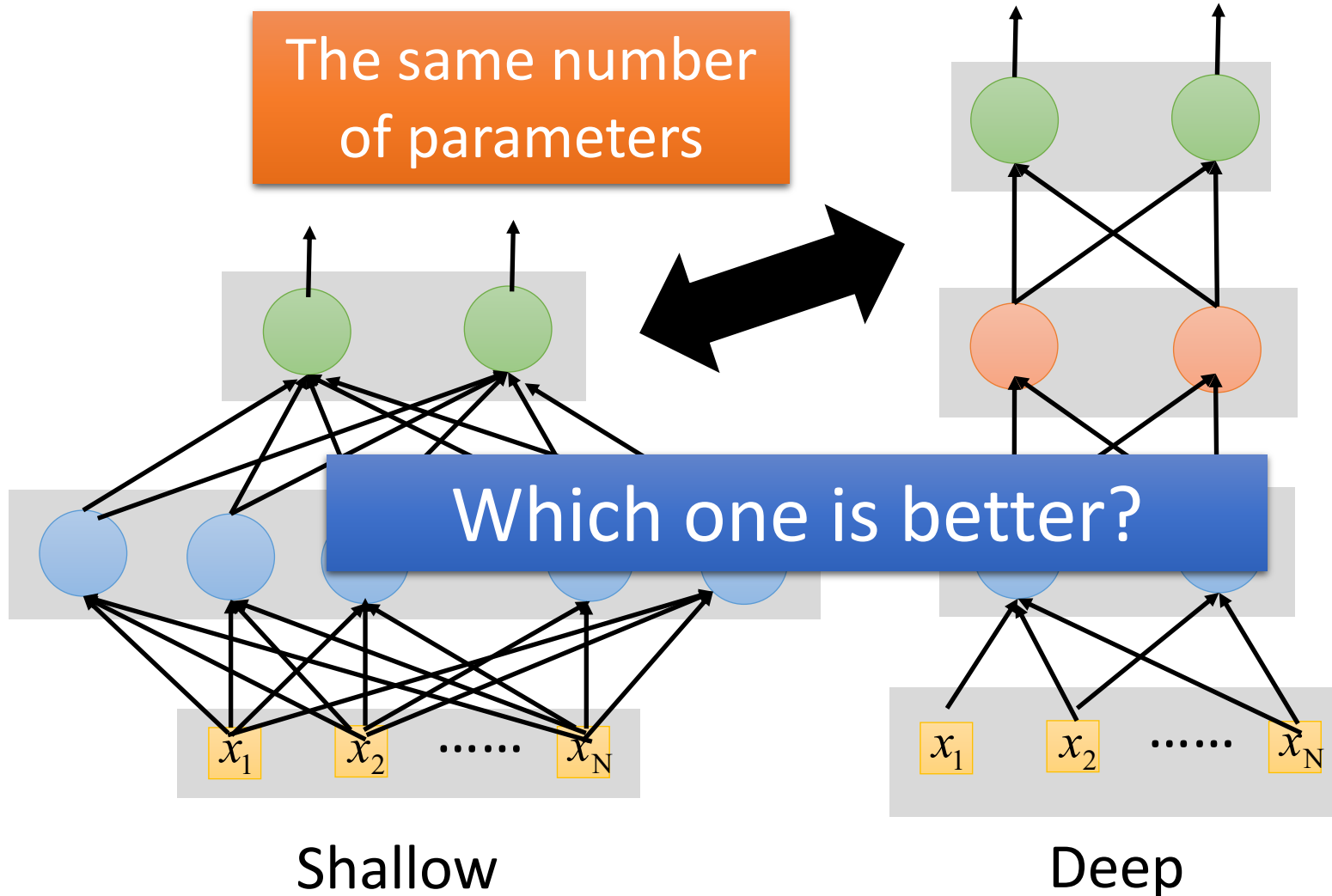
(given **enough** hidden
neurons)



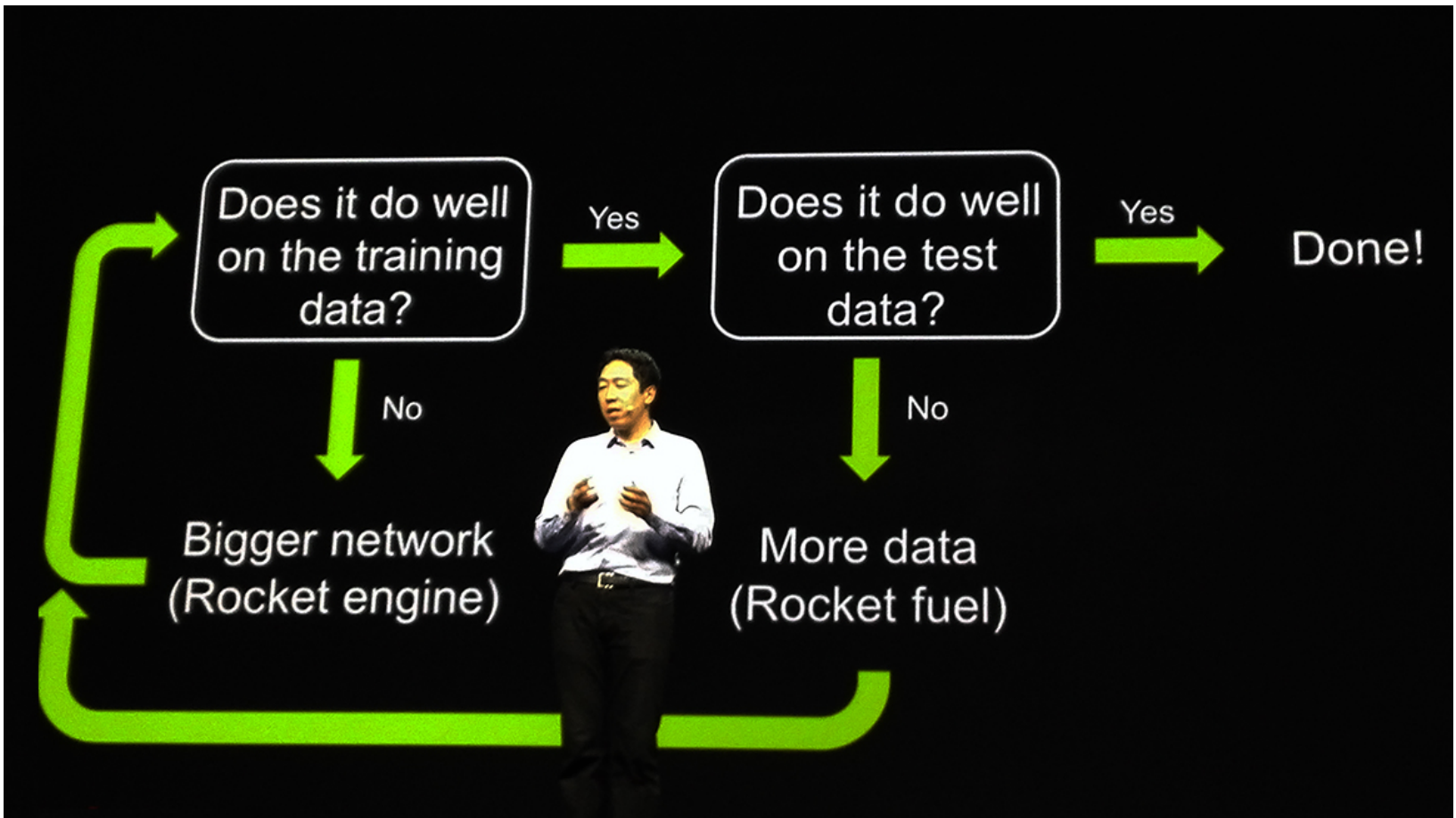
Reference for the reason:
<http://neuralnetworksanddeeplearning.com/chap4.html>

Why “Deep” neural network not “Fat” neural network?

Fat + Short v.s. Thin + Tall

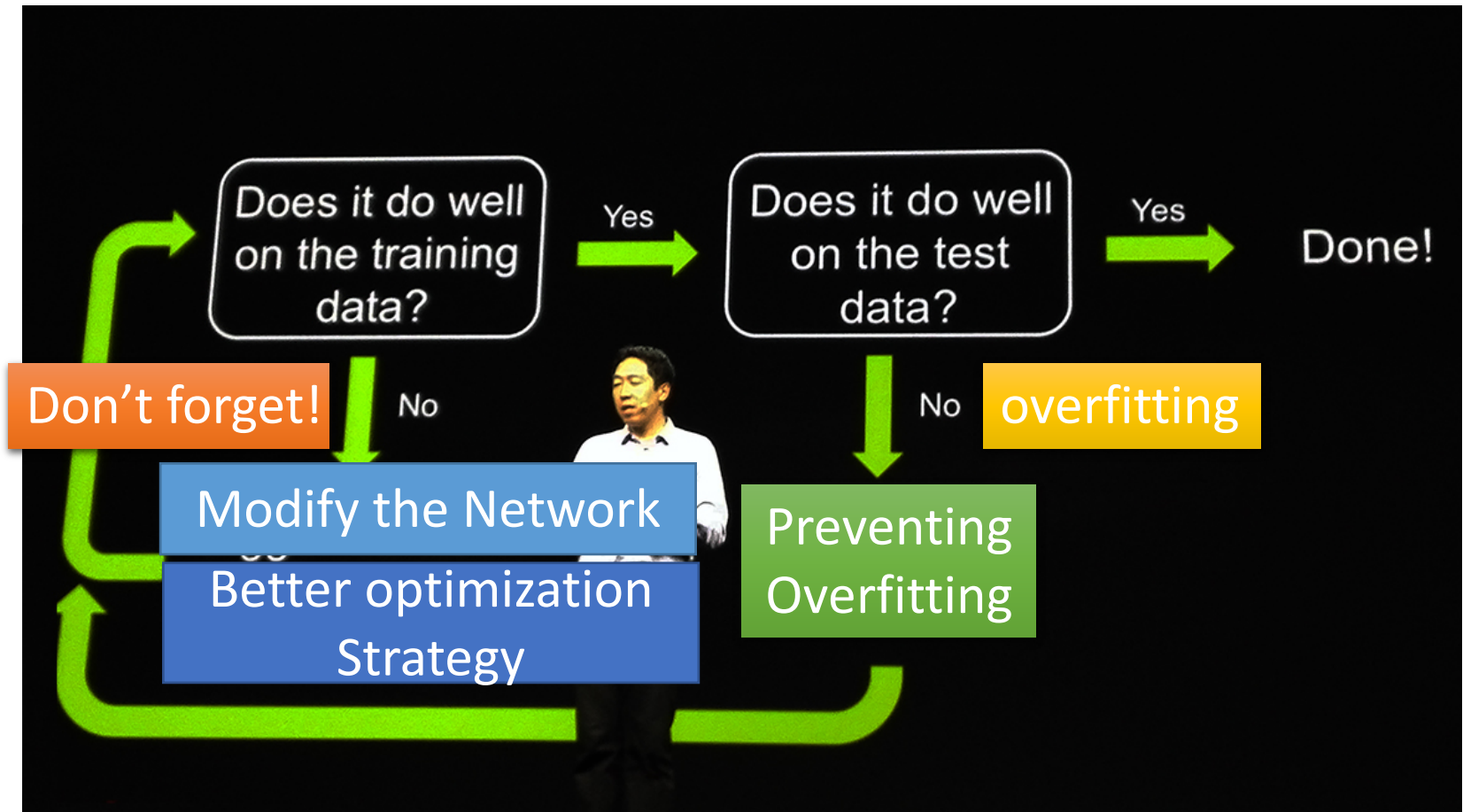


Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Recipe for Learning



<http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/>

Neural networks re-visited

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

Neural networks: without the brain stuff

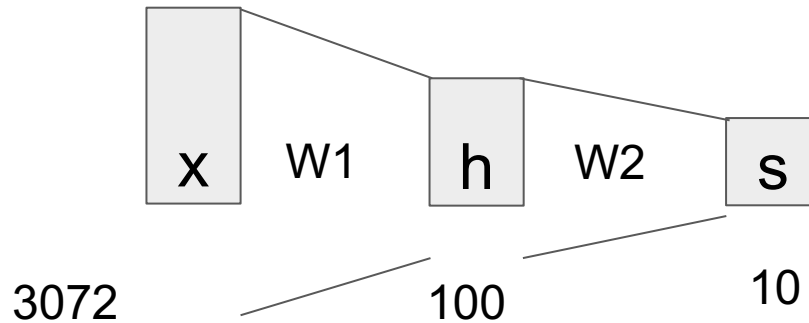
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



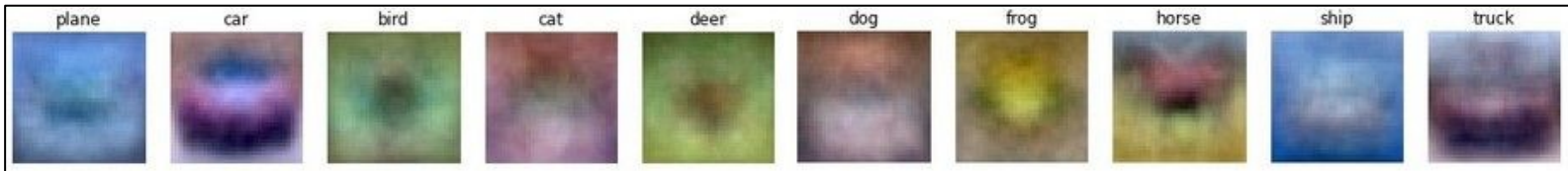
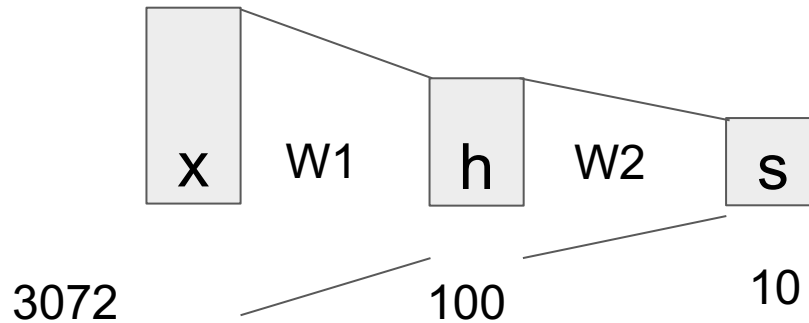
Neural networks: without the brain stuff

(**Before**) Linear score function:

$$f = Wx$$

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

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



Neural networks: without the brain stuff

(Before) Linear score function: $f = Wx$

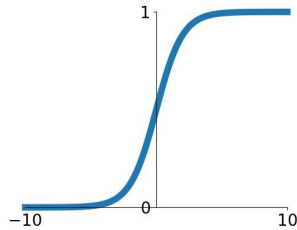
(Now) 2-layer Neural Network
or 3-layer Neural Network $f = W_2 \max(0, W_1 x)$

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Activation functions

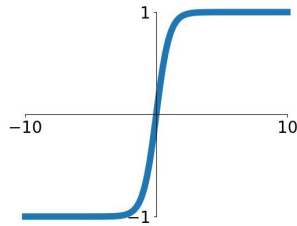
Sigmoid

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



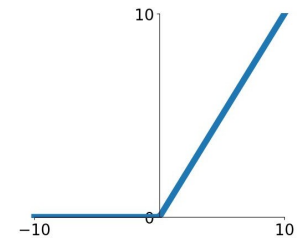
tanh

$$\tanh(x)$$



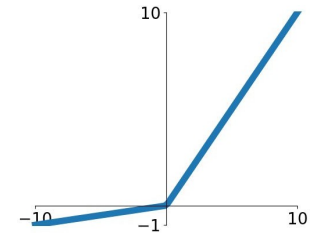
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

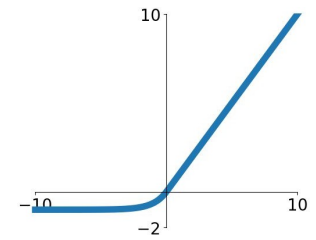


Maxout

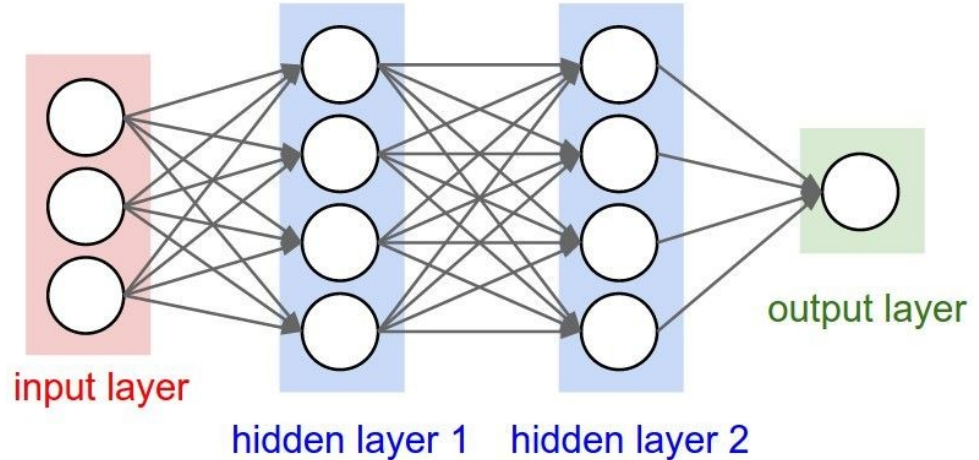
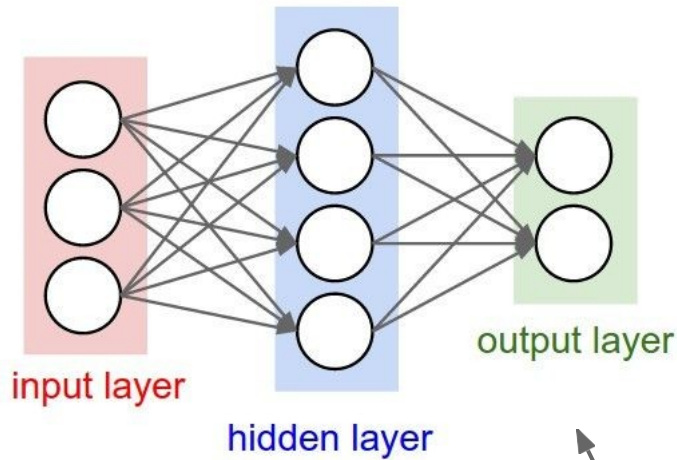
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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



Neural networks: Architectures



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

Next: Convolutional Neural Networks

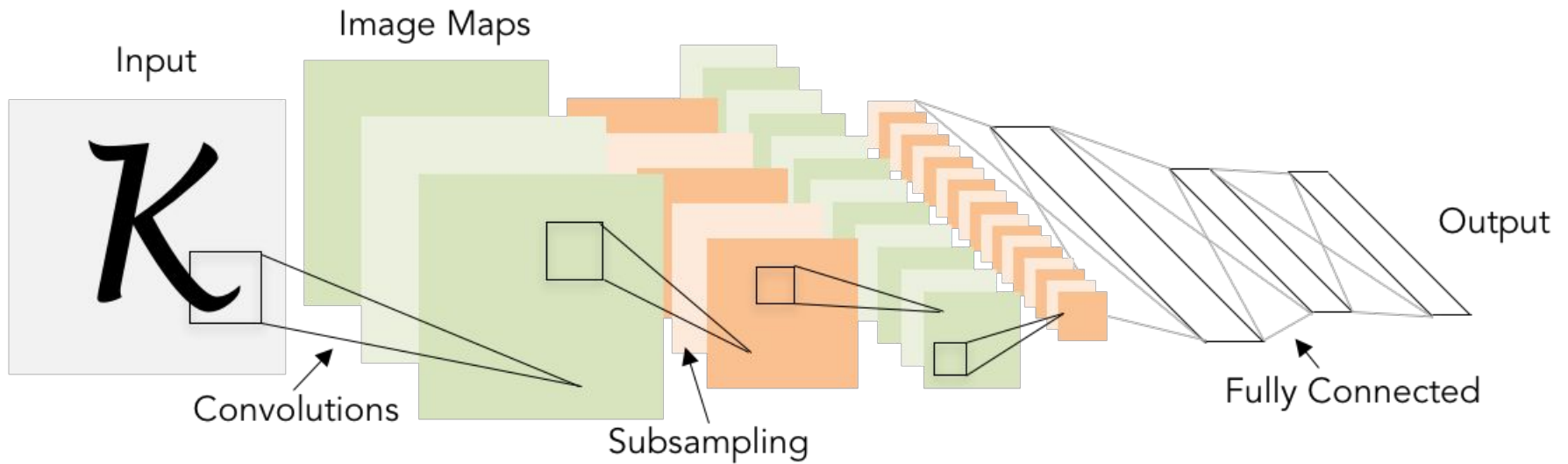
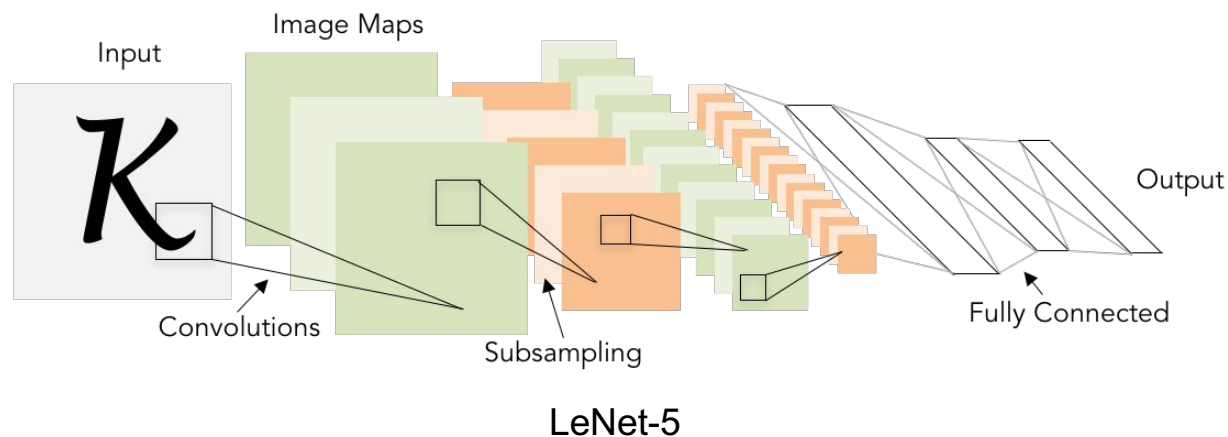


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

A bit of history:



A bit of history: ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

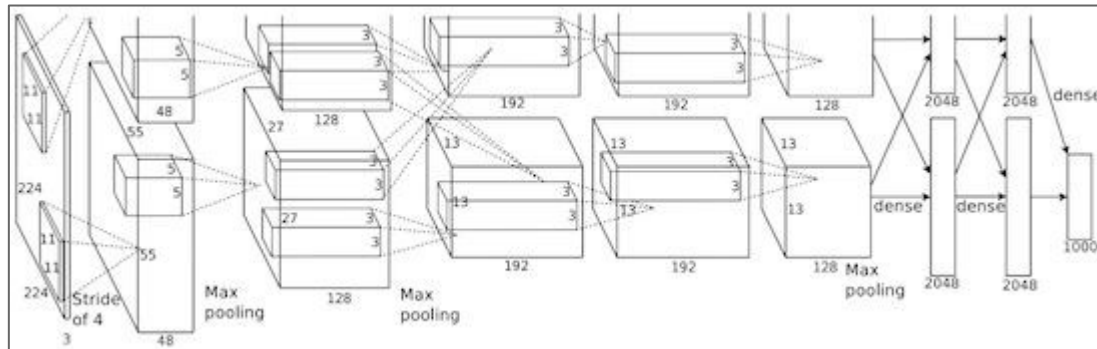


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Fast-forward to today: ConvNets are everywhere



Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars



[This image](#) by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

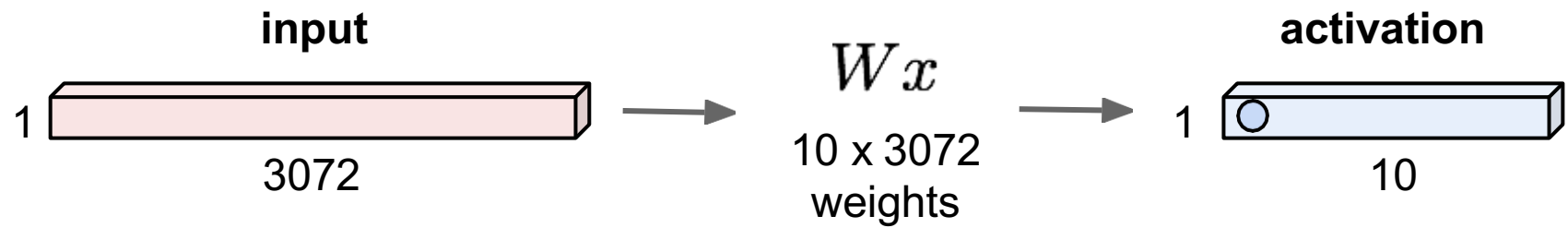
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Convolutional Neural Networks

(First without the brain stuff)

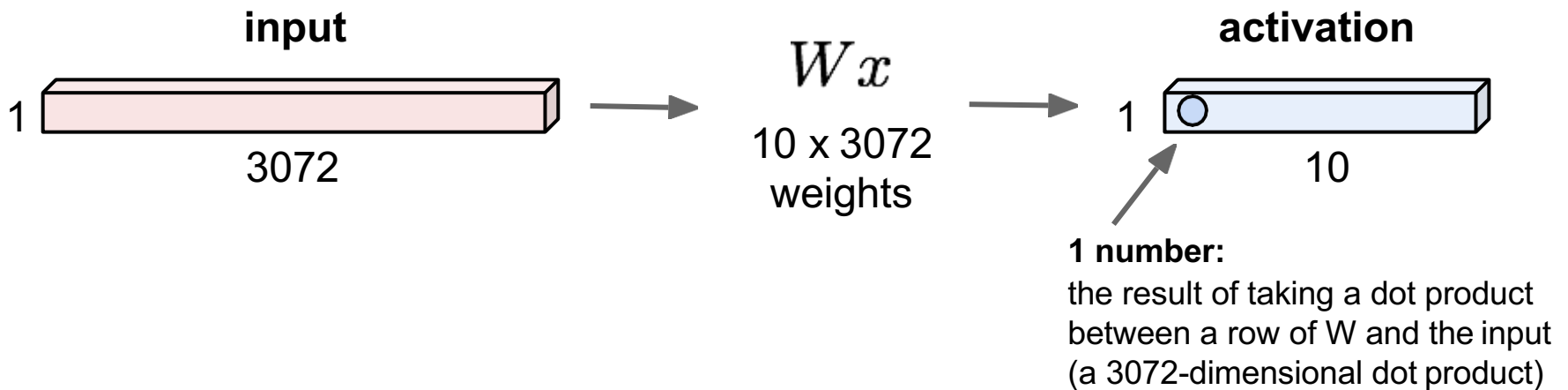
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



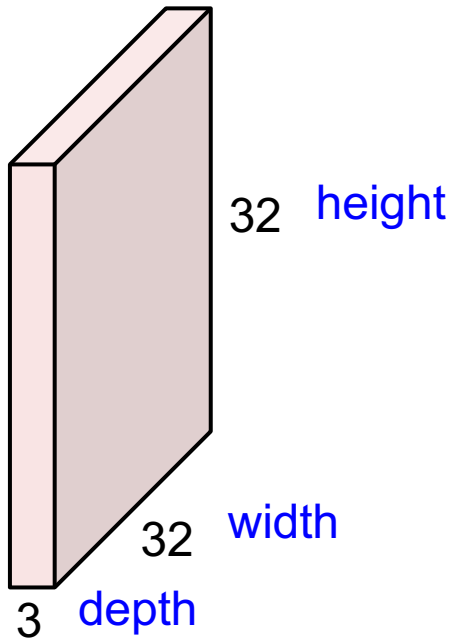
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



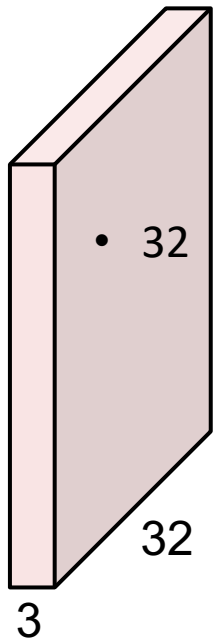
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

- 32x32x3 image



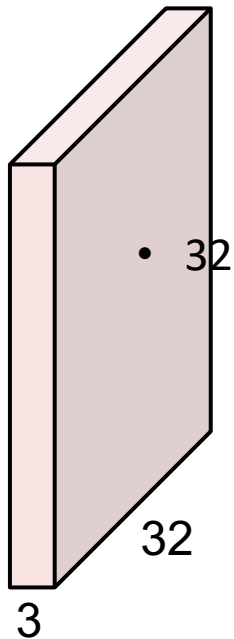
- 5x5x3 filter



- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

- 32x32x3 image



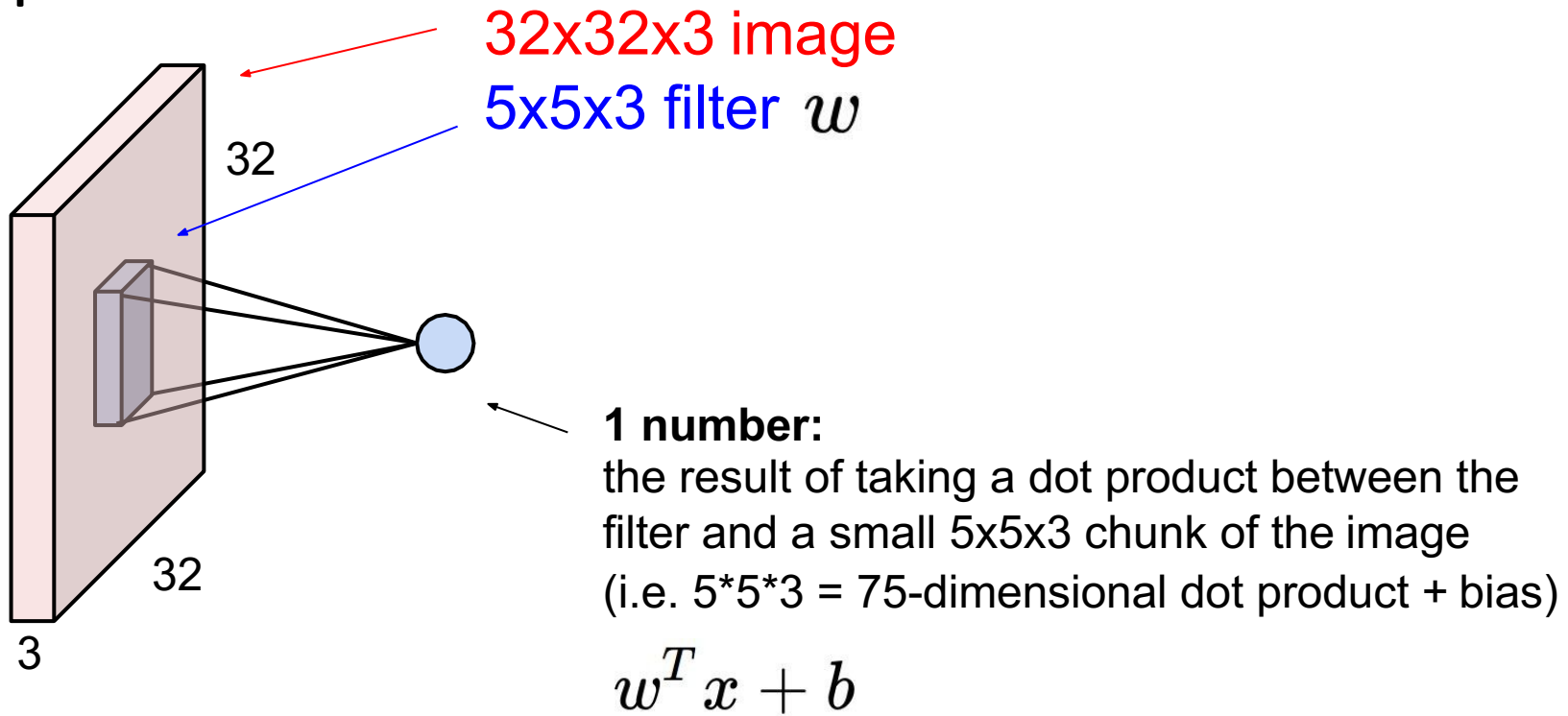
Filters always extend the full depth of the input volume

- 5x5x3 filter

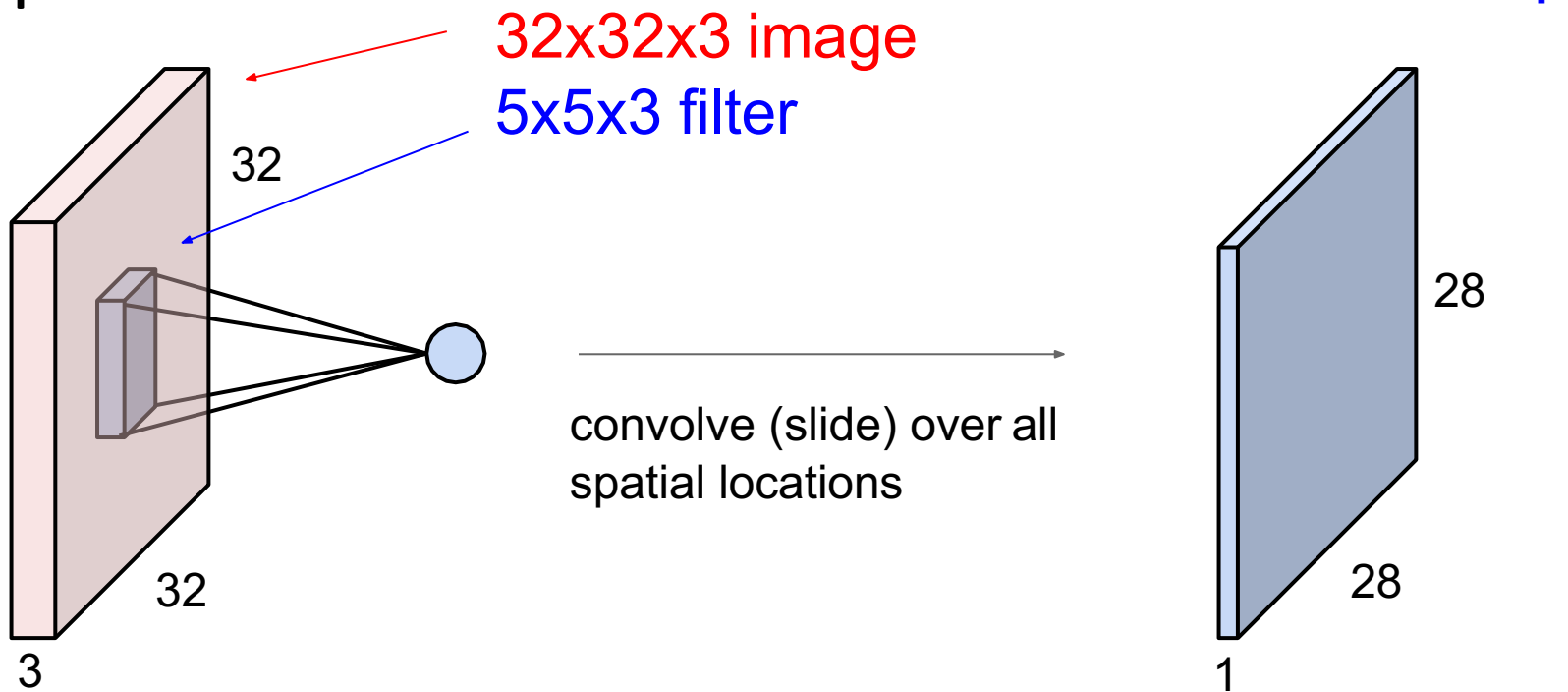


- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

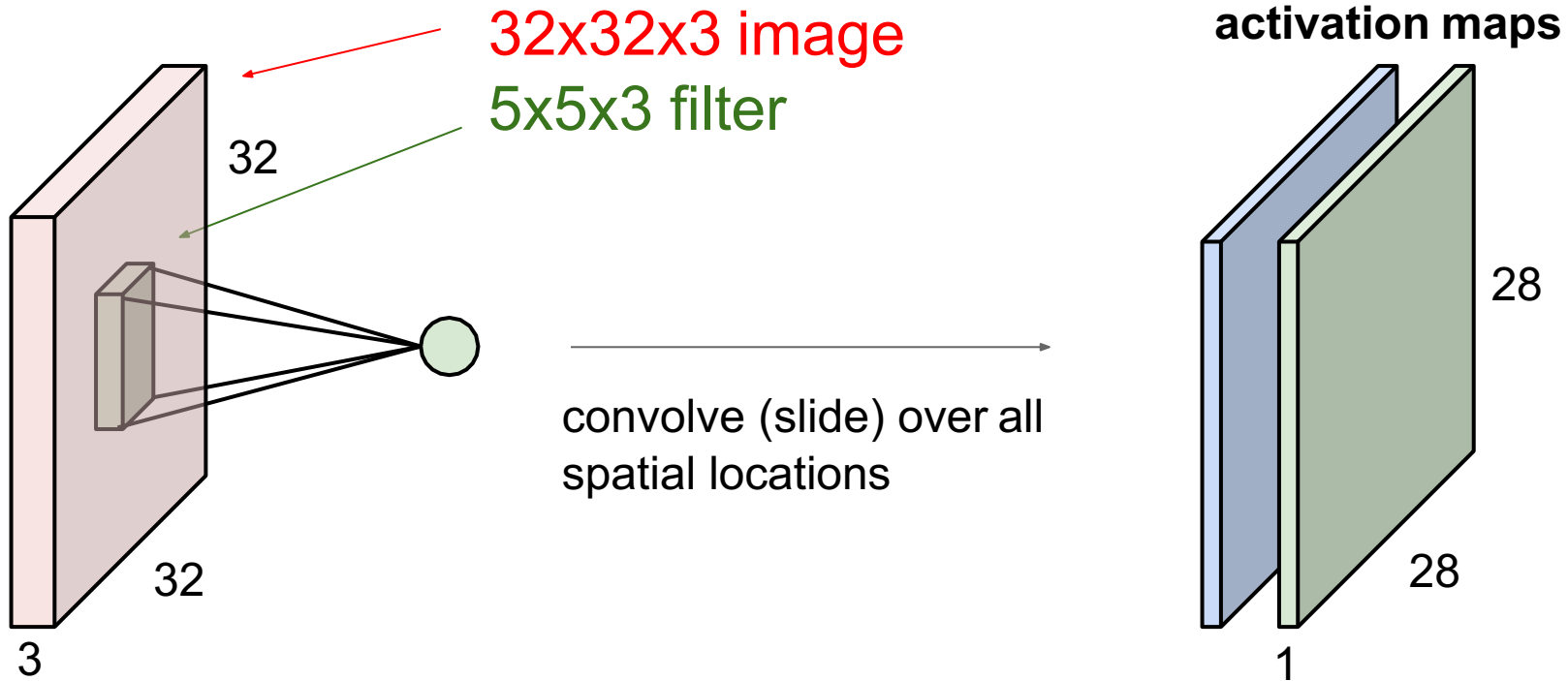


Convolution Layer

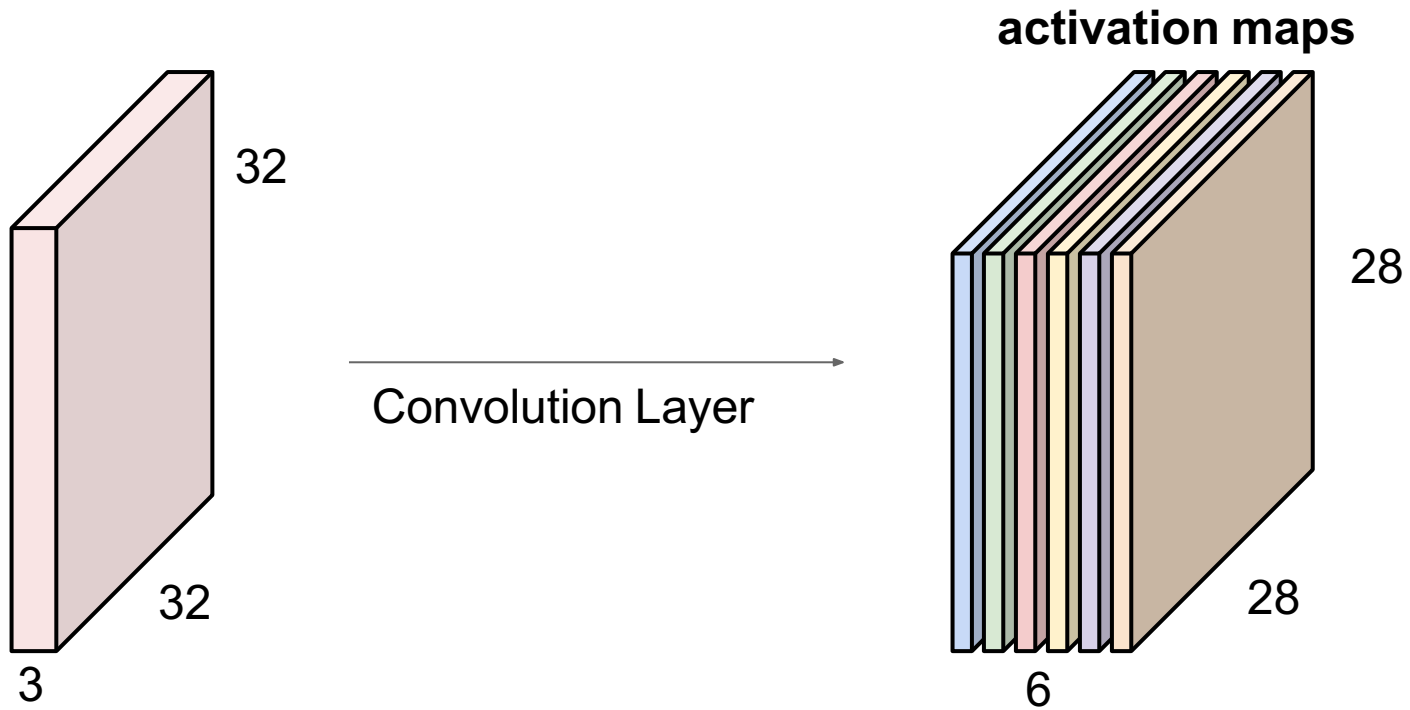


Convolution Layer

consider a second, green filter

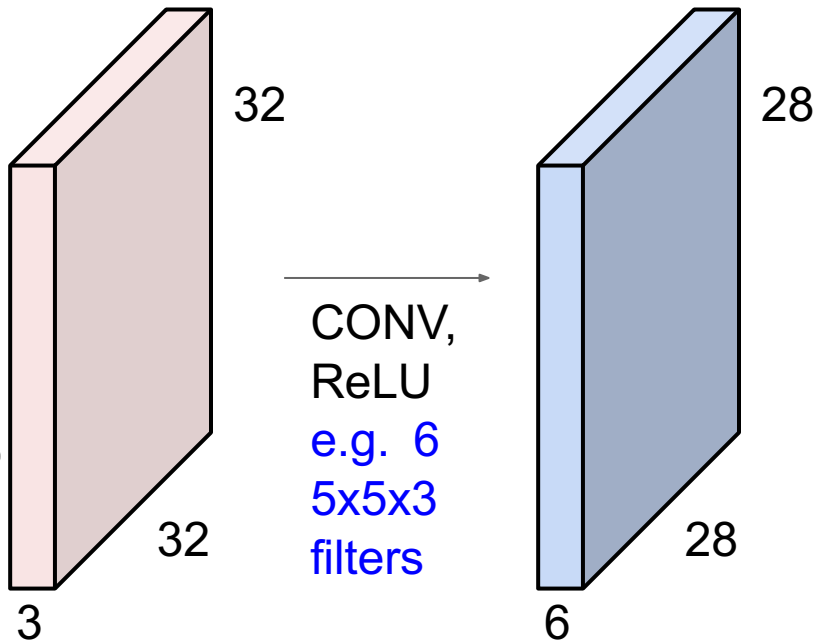


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

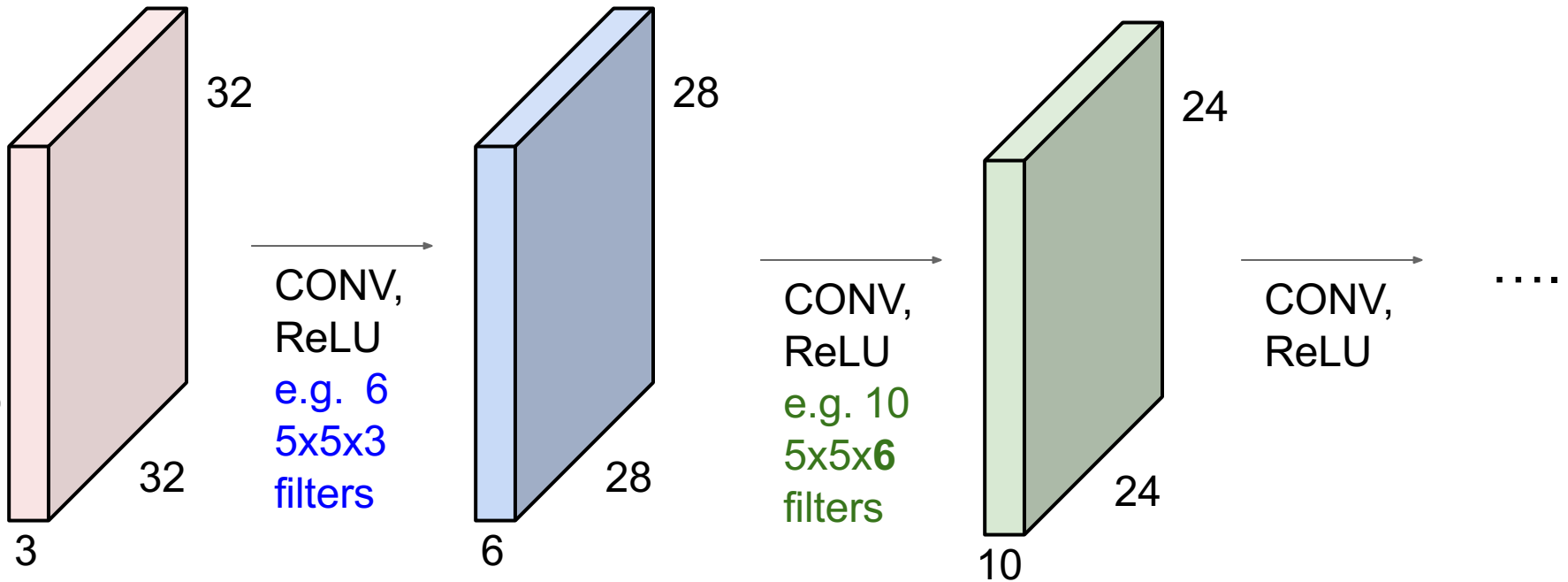


We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



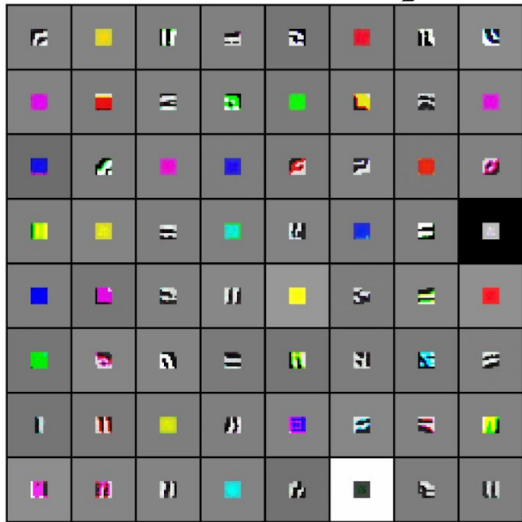
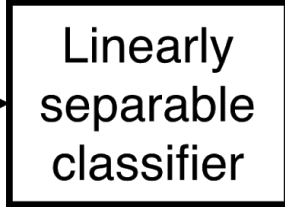
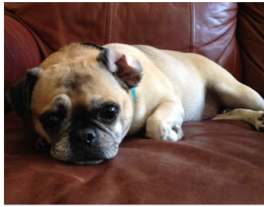
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



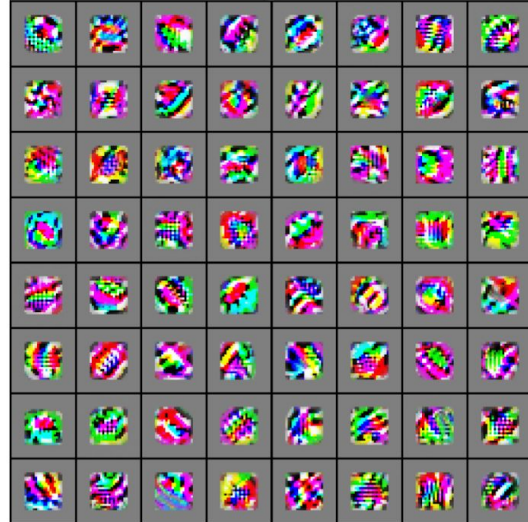
Preview

[Zeiler and Fergus 2013]

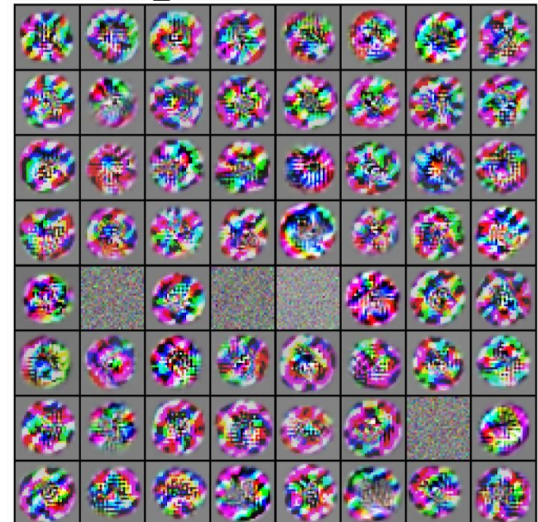
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



VGG-16 Conv1_1

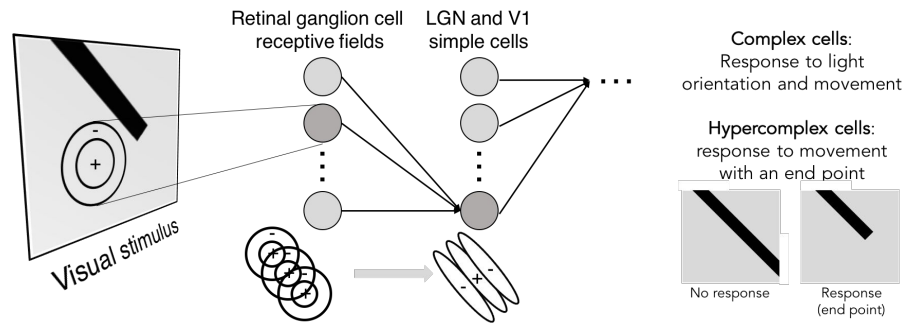
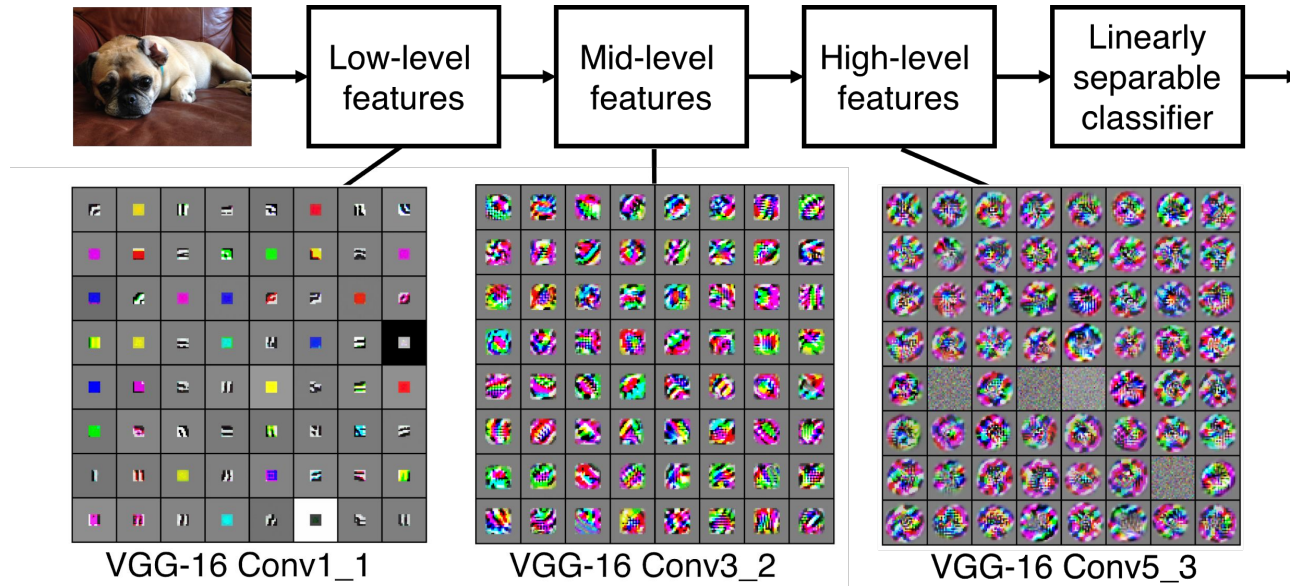


VGG-16 Conv3_2



VGG-16 Conv5_3

Preview





one filter =>
one activation map

example 5x5 filters
(32 total)

Activations:

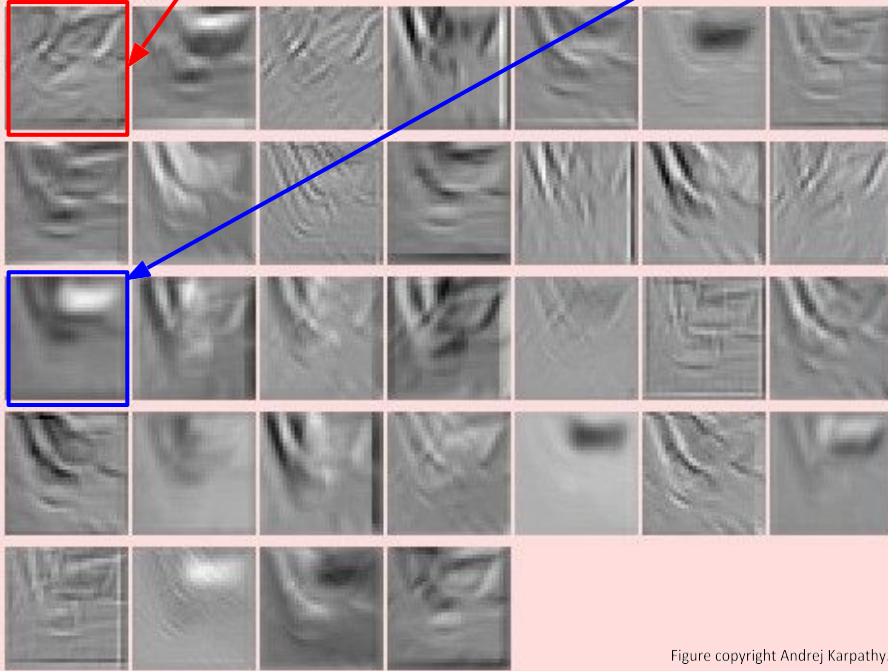


Figure copyright Andrej Karpathy.

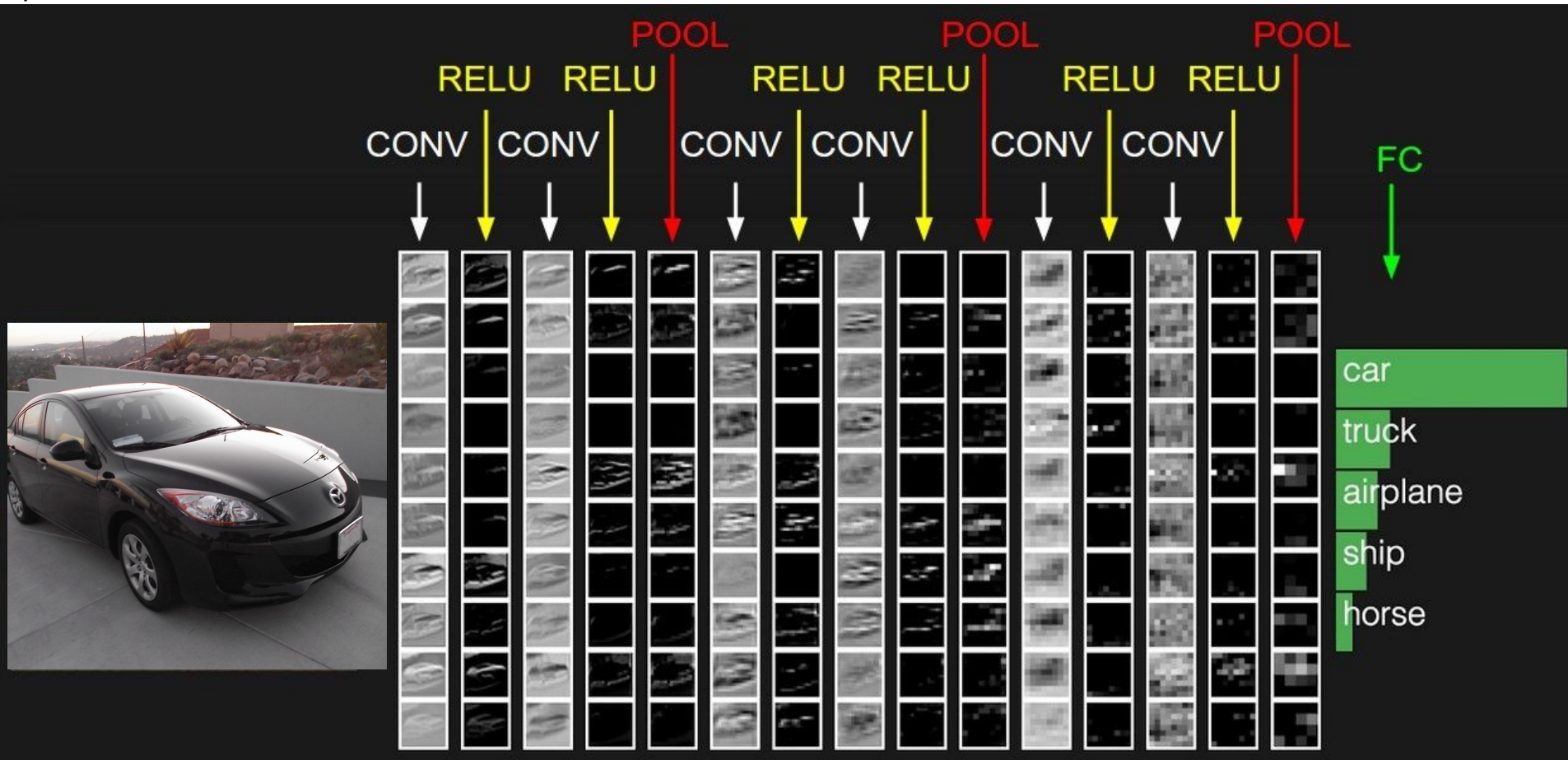
We call the layer convolutional
because it is related to convolution
of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$

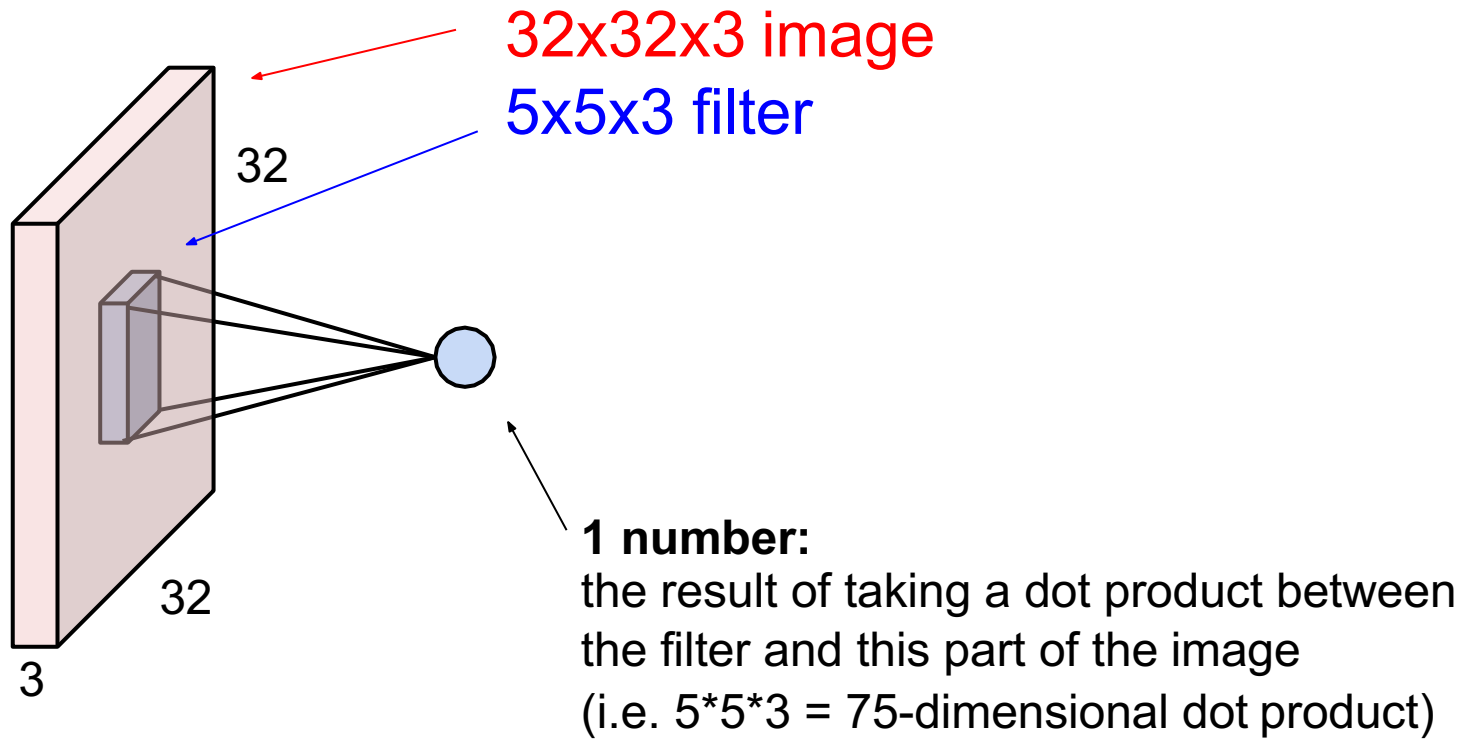


elementwise multiplication and sum of
a filter and the signal (image)

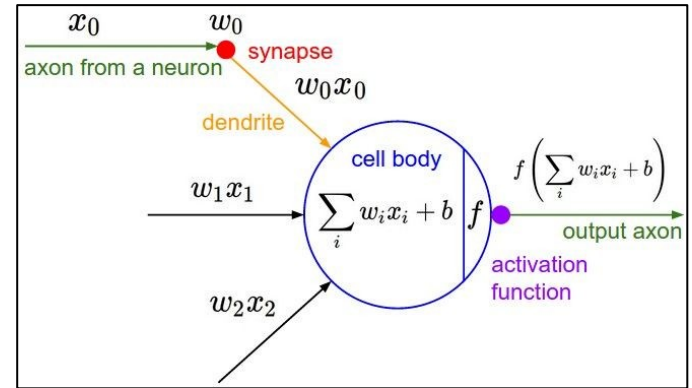
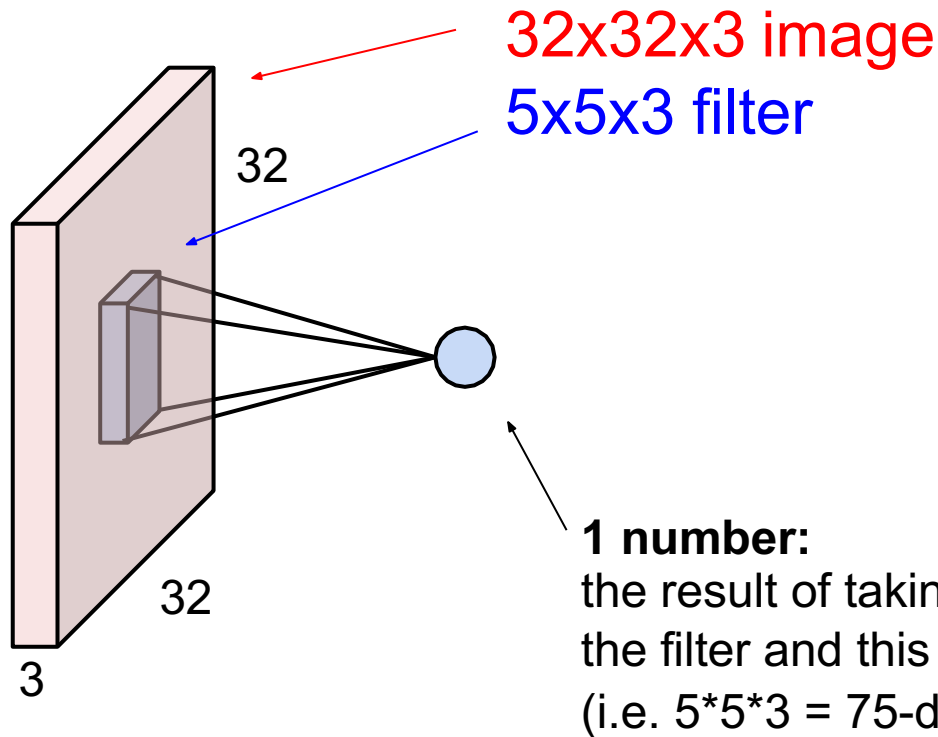
preview:



The brain/neuron view of CONV Layer

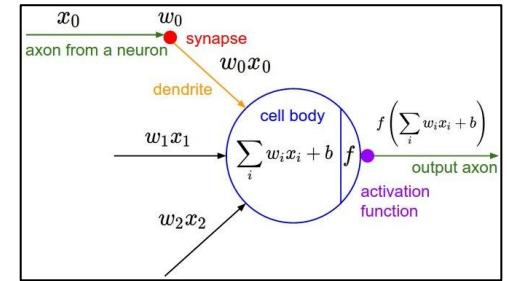
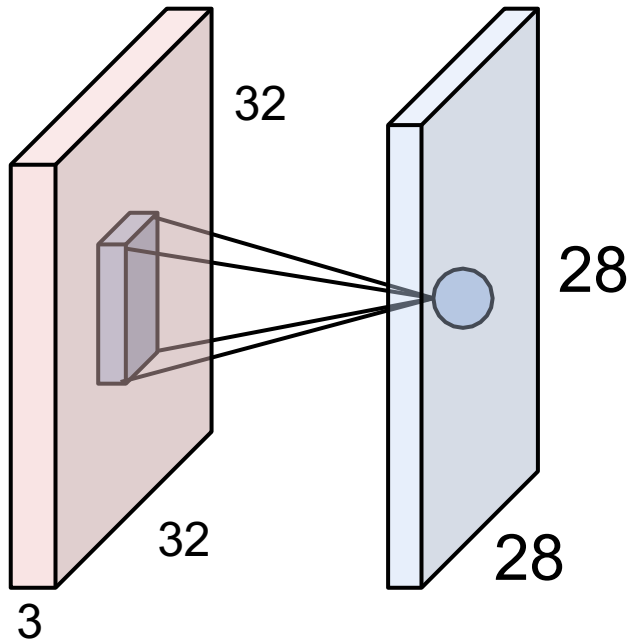


The brain/neuron view of CONV Layer



It's just a neuron with local connectivity...

The brain/neuron view of CONV Layer

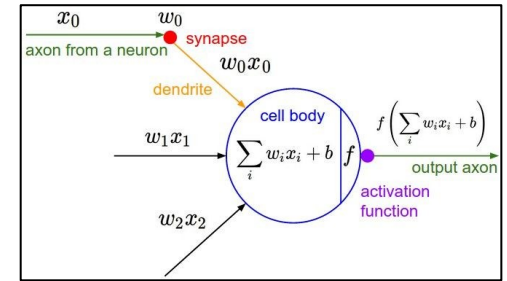
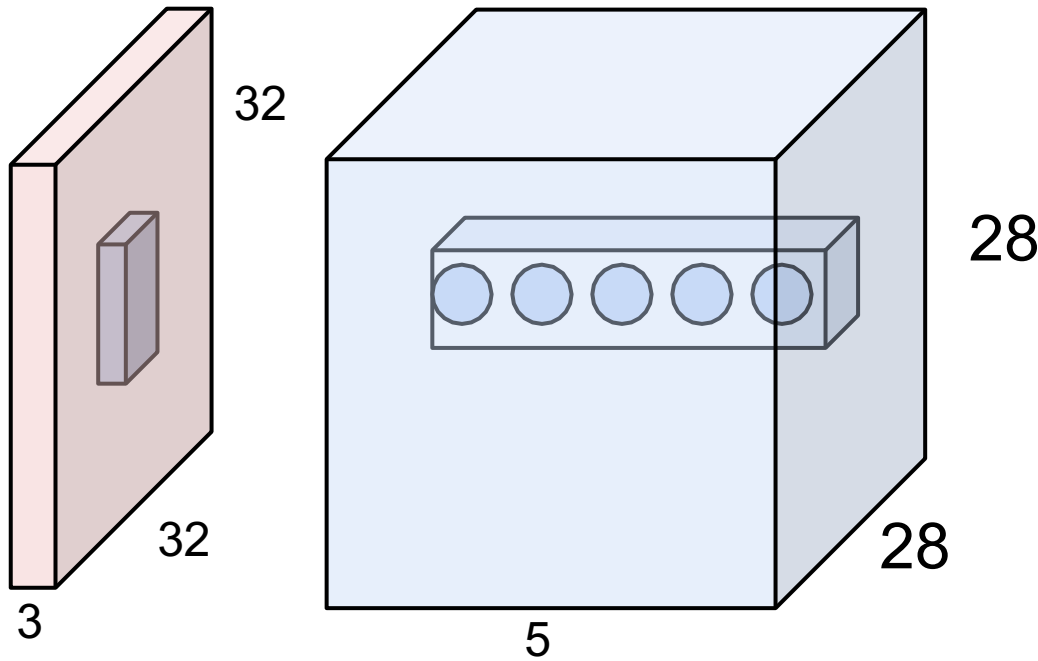


An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

The brain/neuron view of CONV Layer

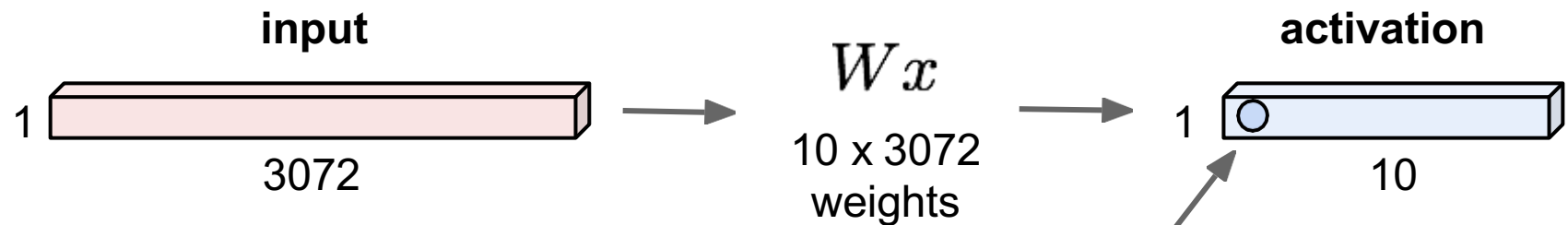


E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

There will be 5 different
neurons all looking at the same
region in the input volume

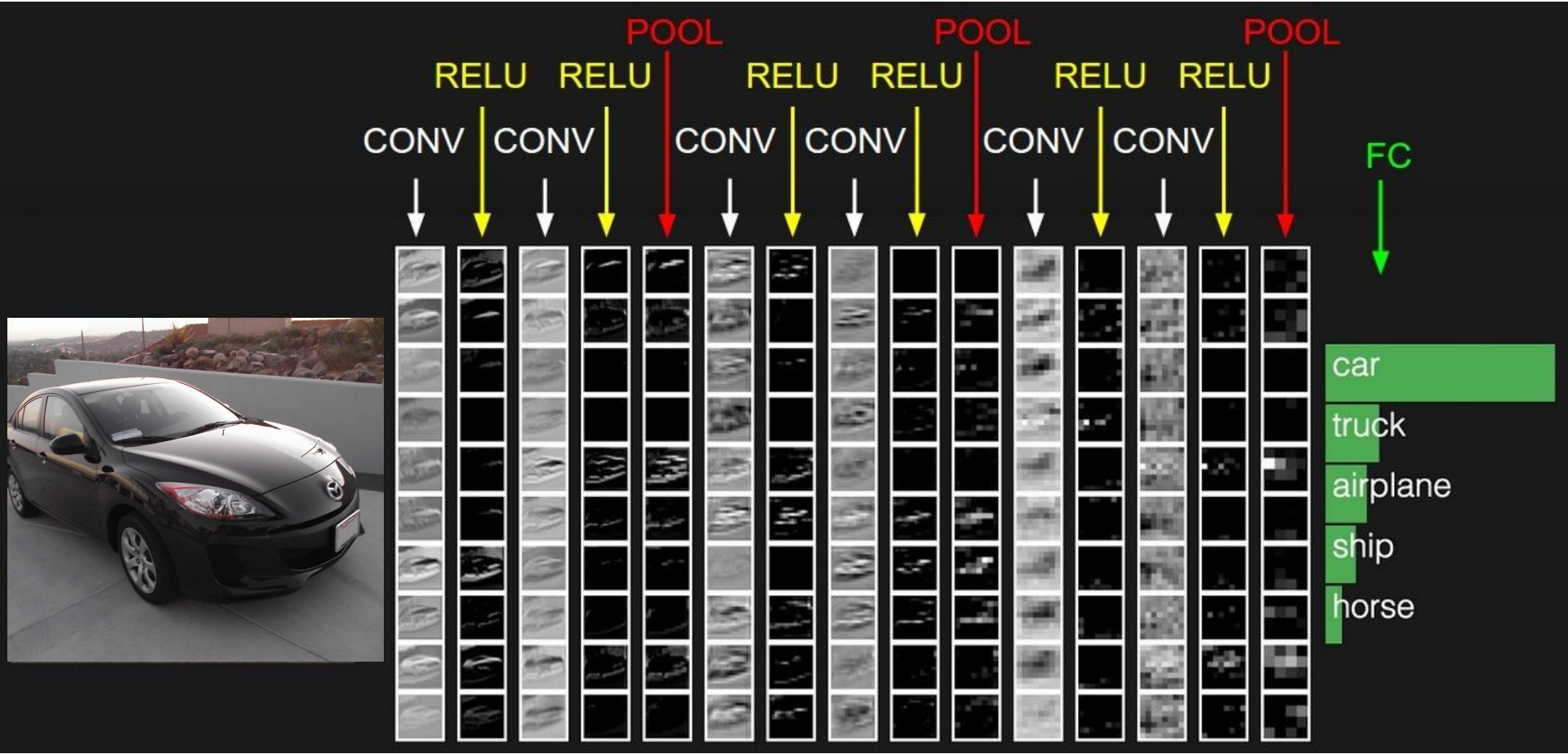
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



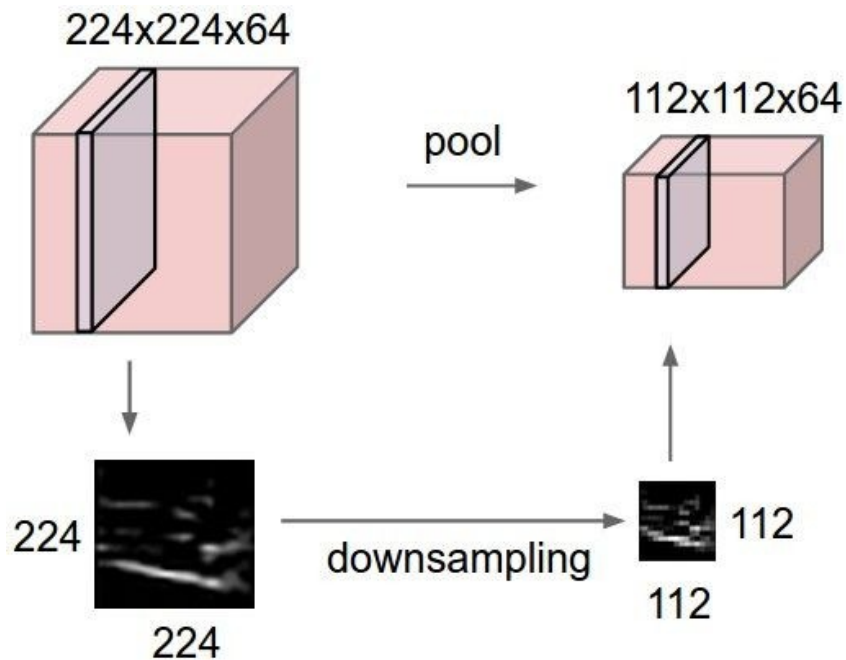
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

two more layers to go: POOL/FC



Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING

Single depth slice

x ↑

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→ y

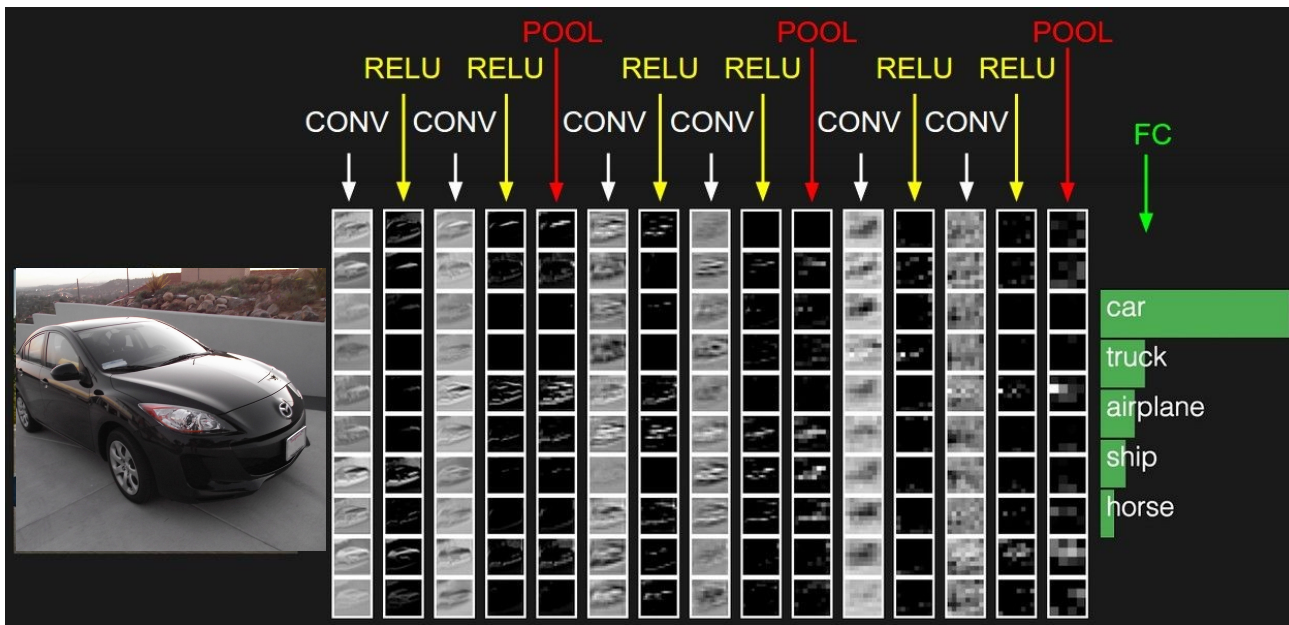
max pool with 2x2 filters
and stride 2



6	8
3	4

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K, SOFTMAX
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet challenge this paradigm