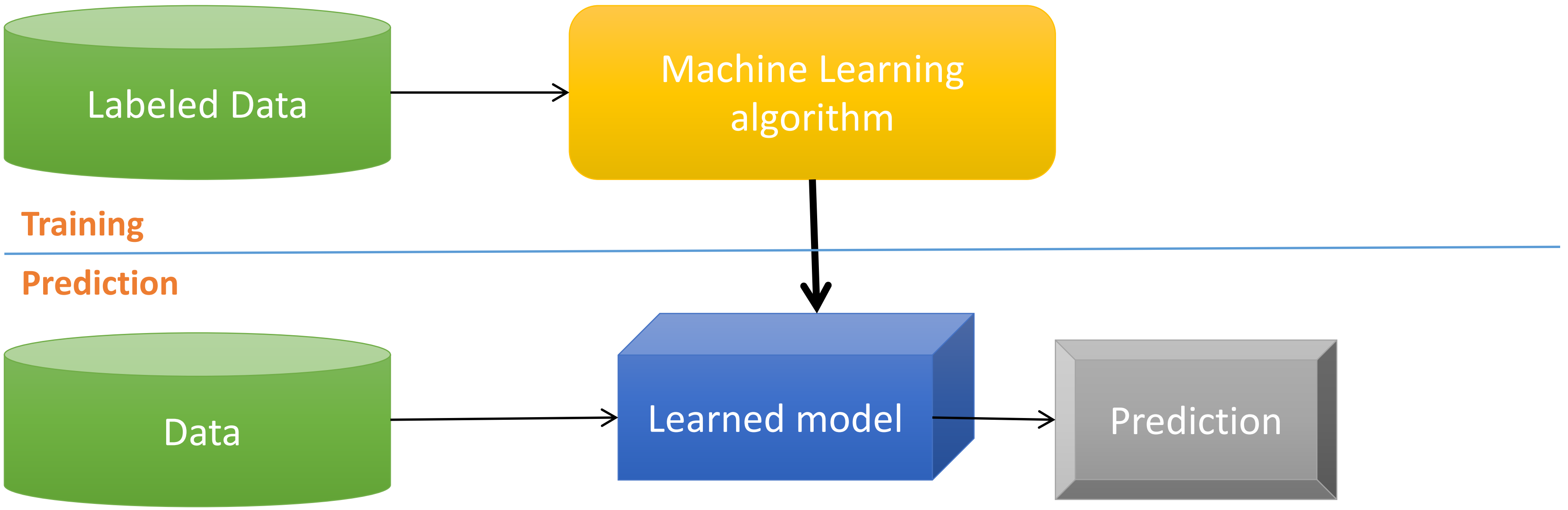
# Deep Learning Tutorial

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**



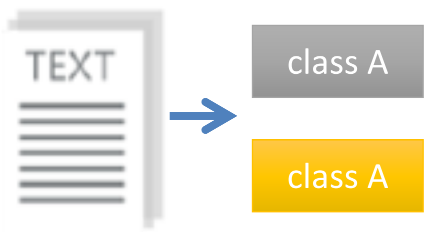

Methods that can learn from and make predictions on data

# Types of Learning

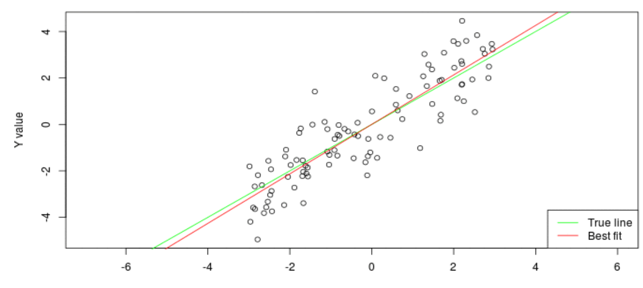
**Supervised**: Learning with a **labeled training** set
Example: email *classification* with already labeled emails

**Unsupervised**: Discover **patterns** in **unlabeled** data
Example: *cluster* similar documents based on text

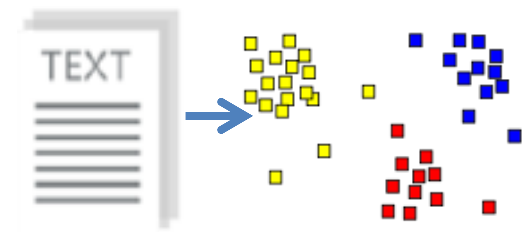
**Reinforcement learning**: learn to **act** based on **feedback/reward**
Example: learn to play Go, reward: *win or lose*




Classification




Regression




Clustering

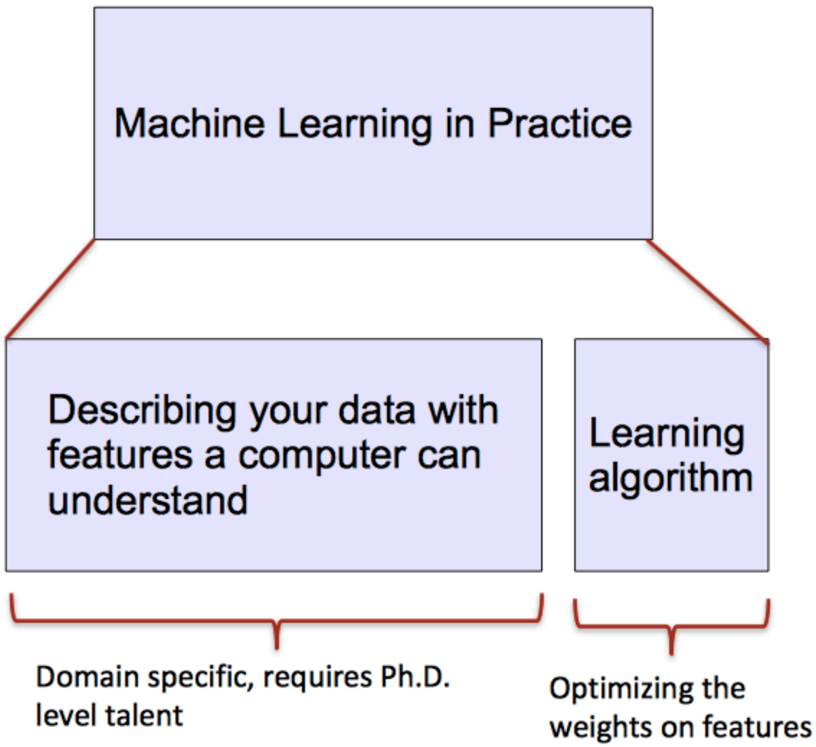
Anomaly Detection
Sequence labeling
…

*http://mbjoseph.github.io/2013/11/27/measure.html*

# ML vs. Deep Learning

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

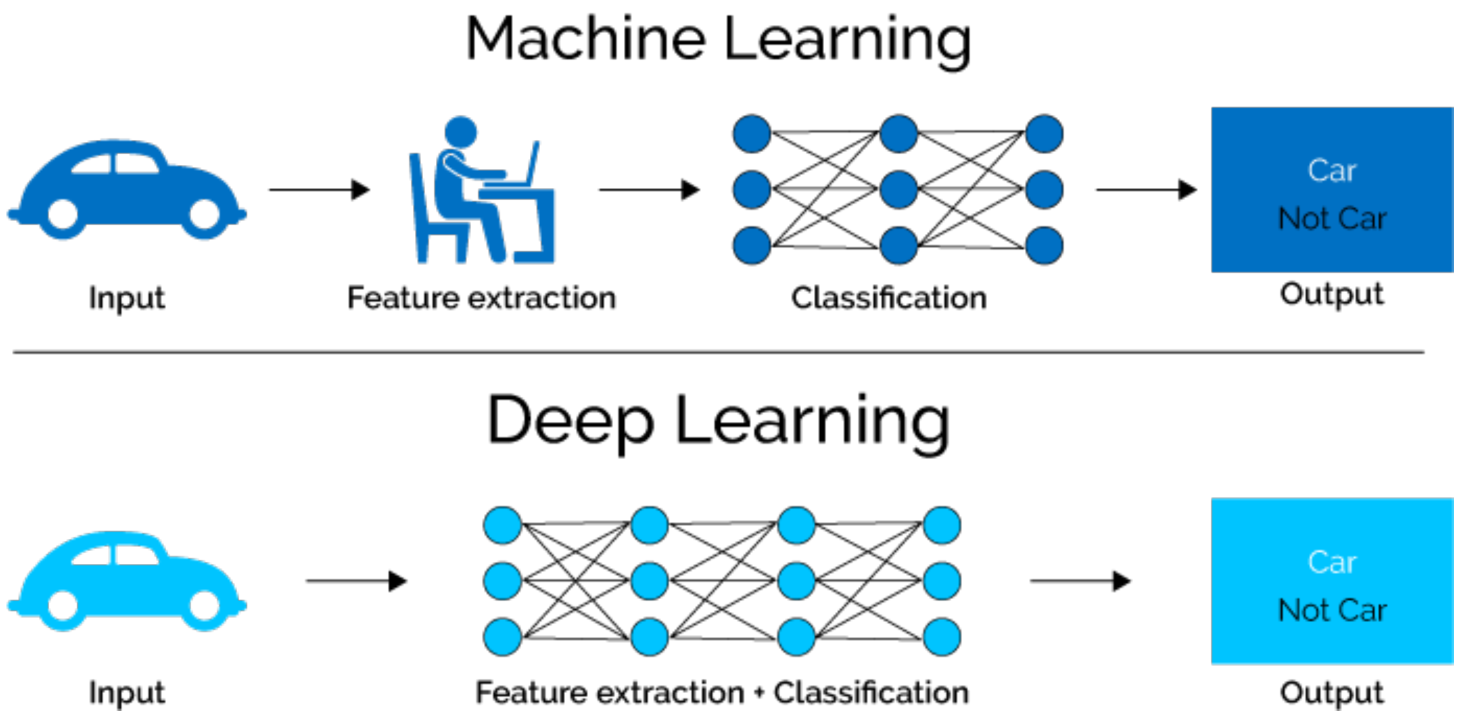
ML becomes just **optimizing weights** to best make a final prediction




| 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 |

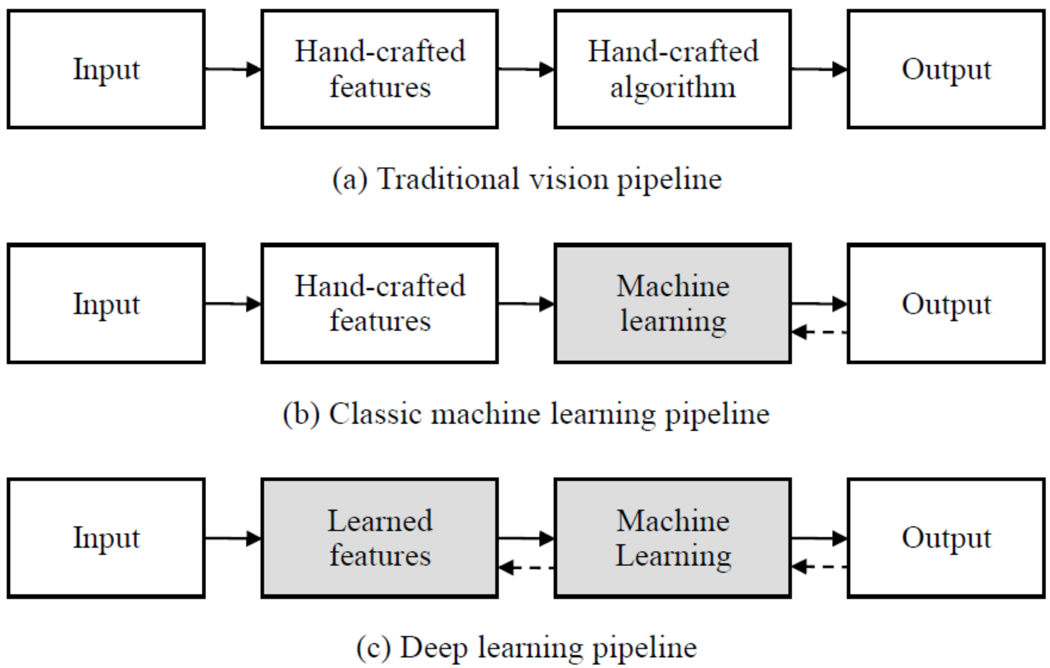
# What is Deep Learning (DL) ?

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.




*https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png*

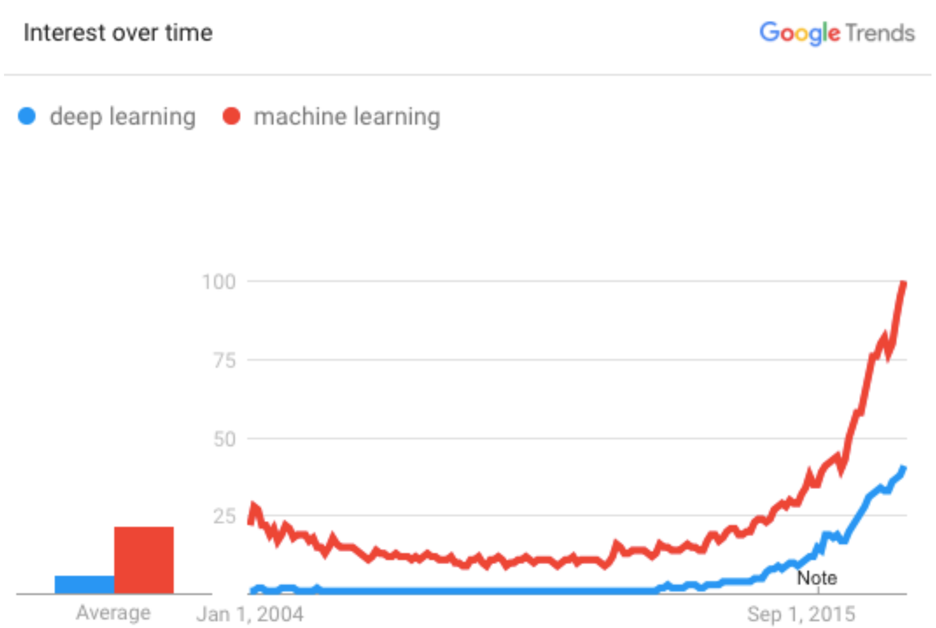
# Traditional and deep learning



# 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

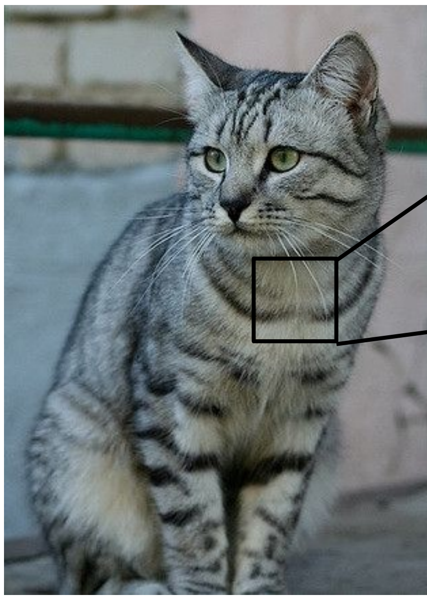


# **Image Classification**: A core task in Computer Vision


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

cat

# **The Problem**: Semantic Gap



This image by Nikita is

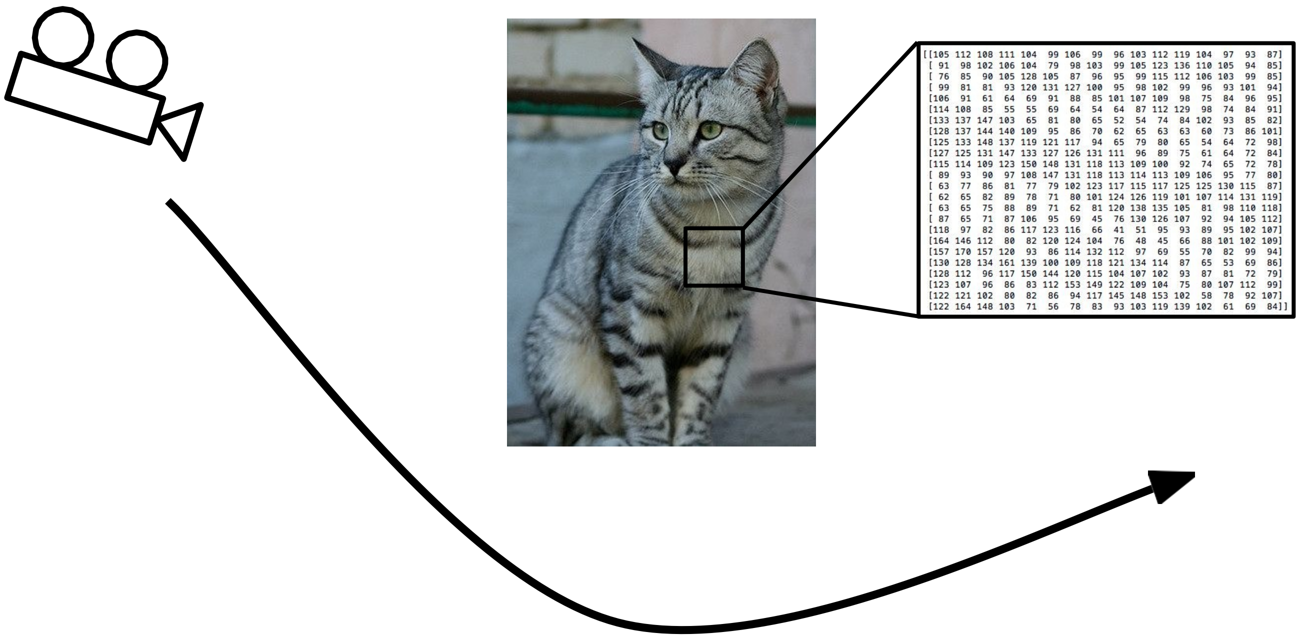

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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  97 108 147 131 118 113 114 113 109 106  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  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)

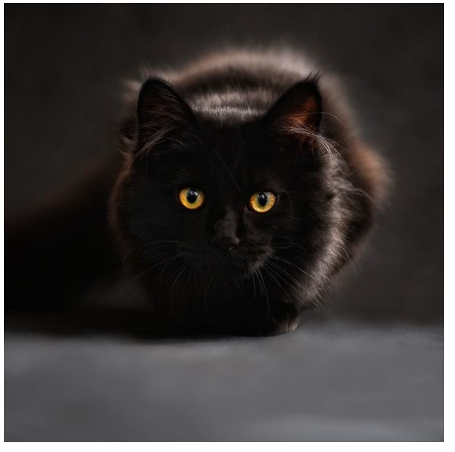
# **Challenges**: Viewpoint variation



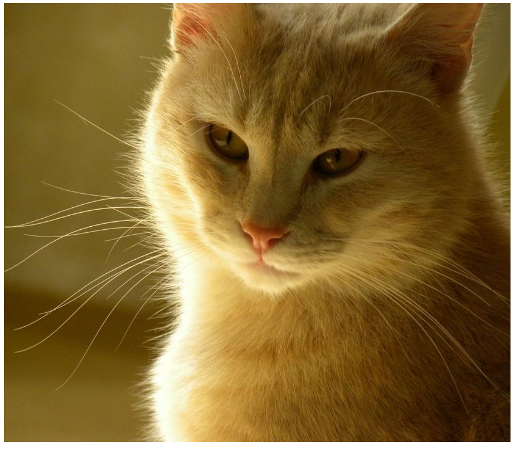

All pixels change when
the camera moves!

# **Challenges**: Illumination



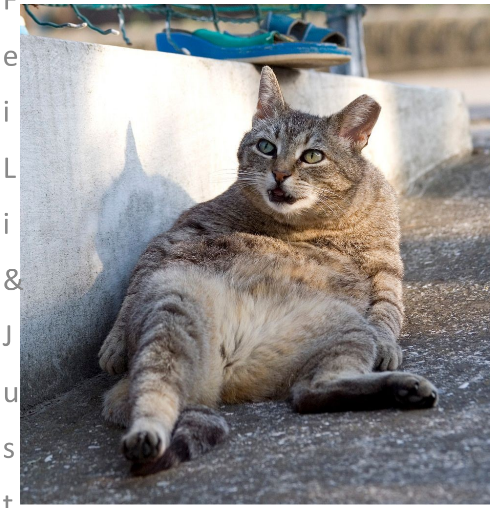








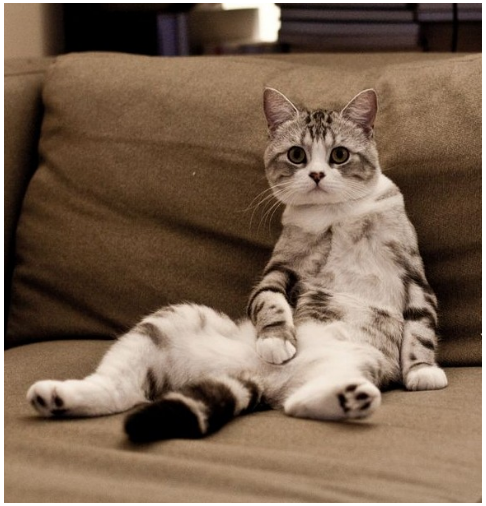
# **Challenges**: Deformation



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

# **Challenges**: Occlusion





# **Challenges**: Background Clutter



# **Challenges**: Intraclass variation

# Linear Classification

# Recall CIFAR10


**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

W

parameters or weights

# Parametric Approach: Linear Classifier

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

Image

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

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

**10** numbers giving
class scores

W

parameters
or weights

# Parametric Approach: Linear Classifier

**3072x1**

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

**10x1** **10x3072**

Image

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

$f($**x**$,$**W**$)$

**10** numbers giving
class scores

W

parameters
or weights

# Parametric Approach: Linear Classifier

3072x1

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

10x1

10x1 10x3072

Image

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

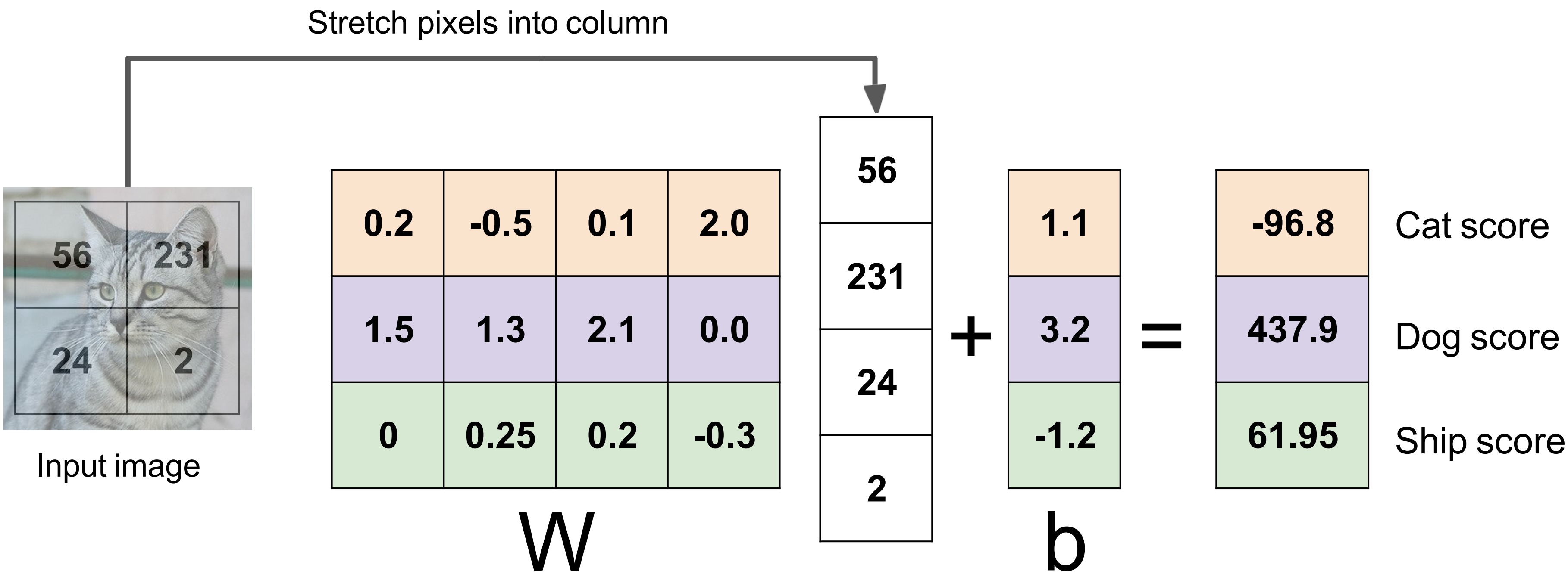
$f(\mathbf{x},\mathbf{W})$

**10** numbers giving class scores

W

parameters or weights

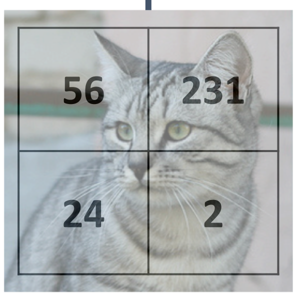
# Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



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

Stretch pixels into column

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




Input image
(2, 2)

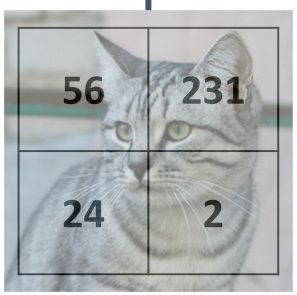
| |
|---|
| **56** |
| **231** |
| **24** |
| **2** |

(4,)

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

Stretch pixels into column

f(x,W) = Wx + b




Input image
(2, 2)

$$\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

W (3, 4)

(4,)

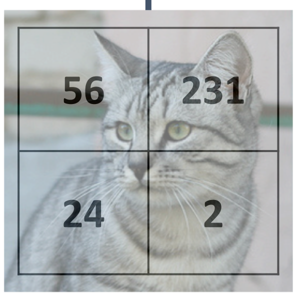
b
(3,)

(3,)

# Linear Classifier: Algebraic Viewpoint

Stretch pixels into column

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



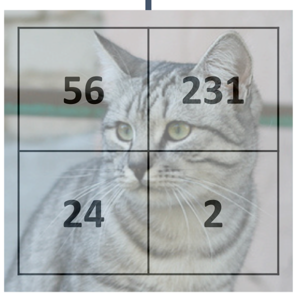

Input image
(2, 2)

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W\ (3,\ 4)} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{(4,)} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b\ (3,)} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{(3,)}$$

# 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)

$$\underset{W\ (3,\ 5)}{\left[\begin{array}{cccc|c} 0.2 & -0.5 & 0.1 & 2.0 & 1.1 \\ 1.5 & 1.3 & 2.1 & 0.0 & 3.2 \\ 0 & 0.25 & 0.2 & -0.3 & -1.2 \end{array}\right]} \underset{(5,)}{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \\ 1 \end{bmatrix}} = \underset{(3,)}{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}$$

# 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!

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

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

Image

Scores

| |
|---|
| -96.8 |
| 437.8 |
| 62.0 |

0.5 * Image

0.5 * Scores

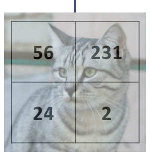
| |
|---|
| -48.4 |
| 218.9 |
| 31.0 |

# Interpreting a Linear Classifier

Algebraic Viewpoint

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

Stretch pixels into column



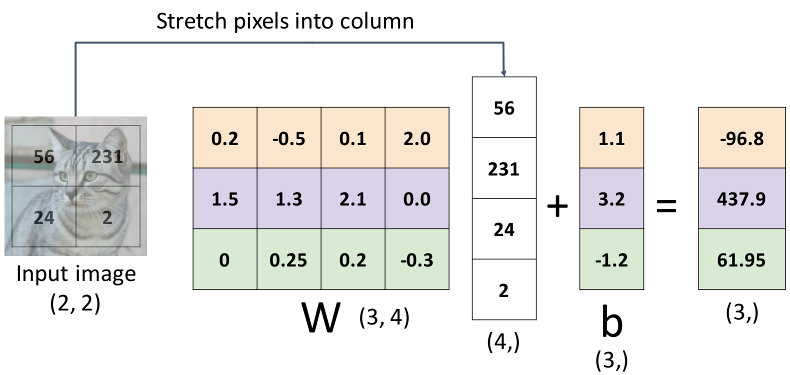

Input image (2, 2)

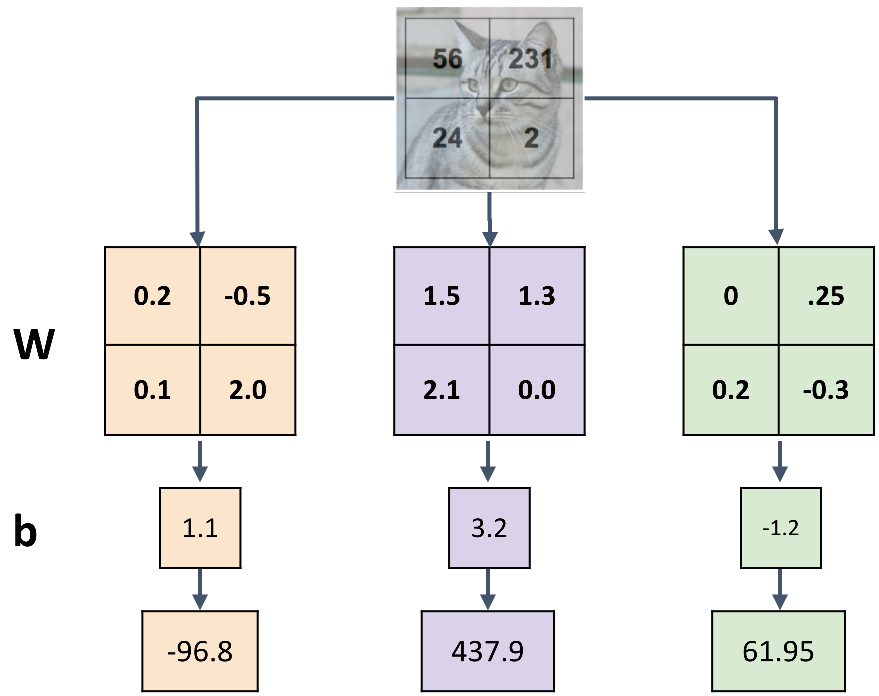
$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W\ (3,4)} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{(4,)} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b\ (3,)} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{(3,)}$$

# Interpreting a Linear Classifier

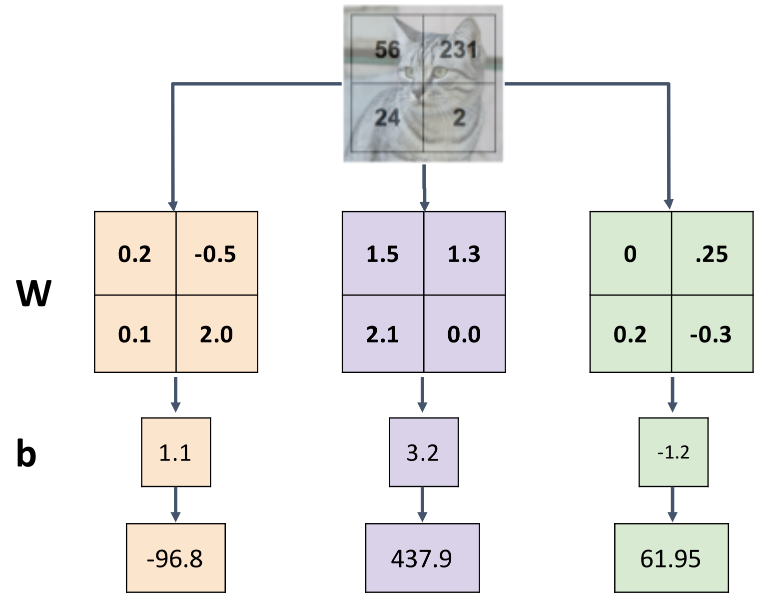
## Algebraic Viewpoint

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

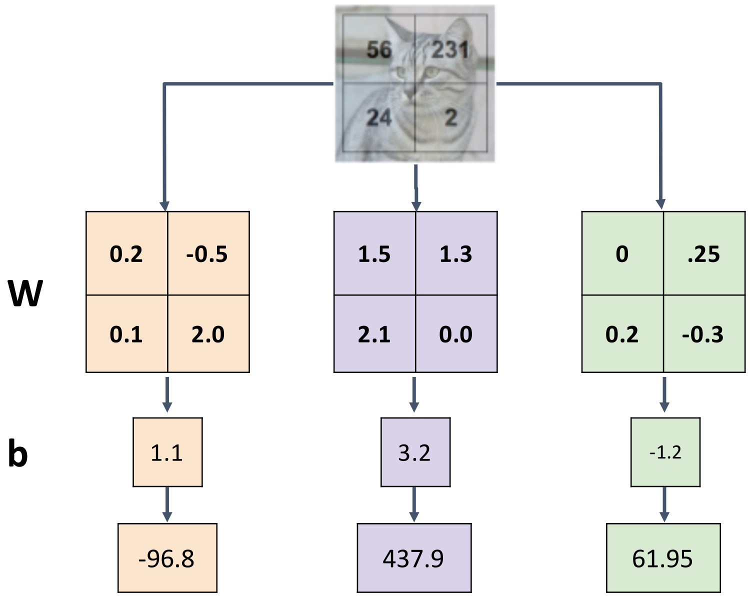


# Interpreting an Linear Classifier

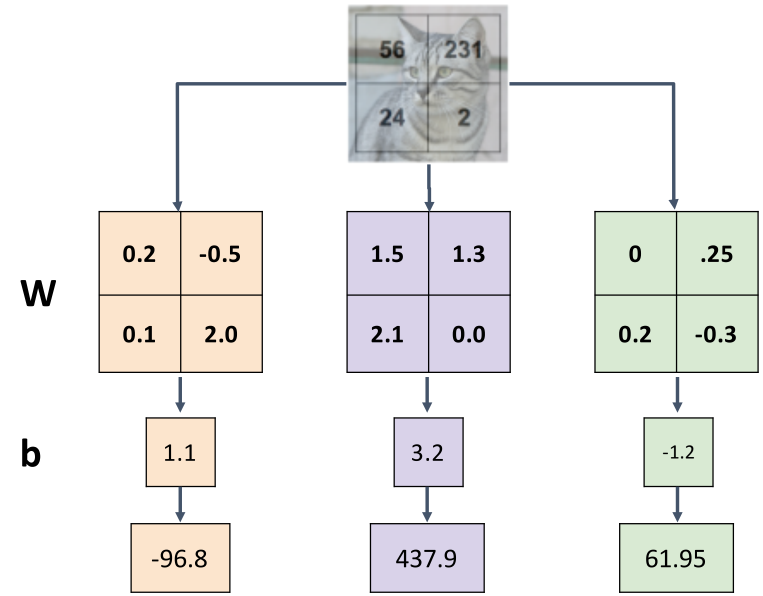



# Interpreting an Linear Classifier: Visual Viewpoint


# Interpreting an Linear Classifier: Visual Viewpoint

Linear classifier has one "template" per category

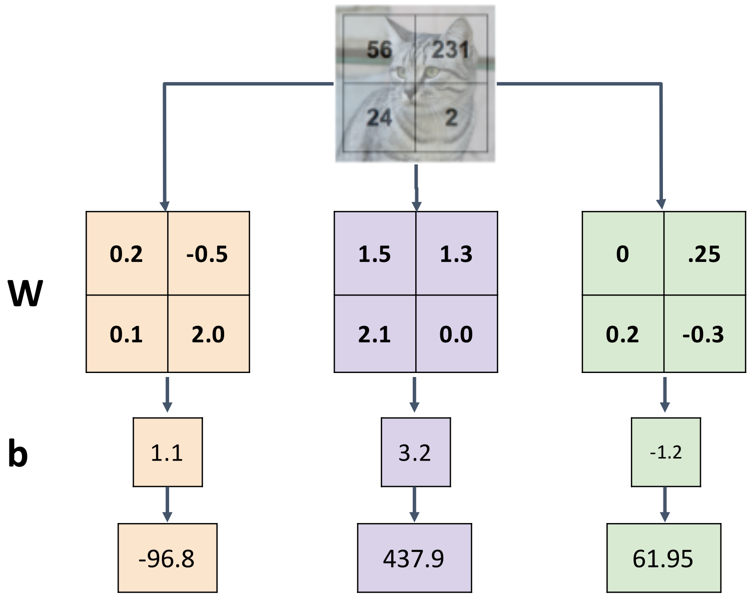



# Interpreting an Linear Classifier: Visual Viewpoint

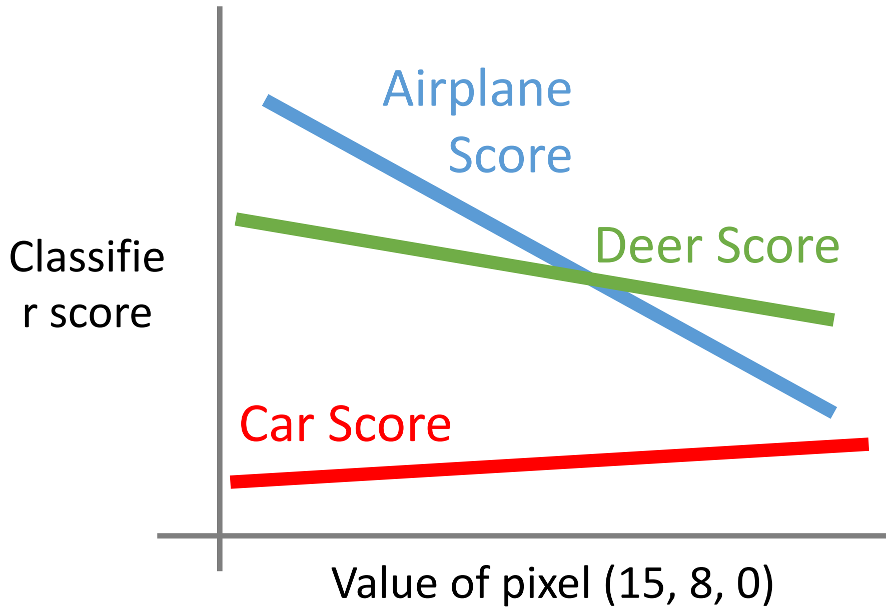
Linear classifier has one "template" per category

A single template cannot capture multiple modes of the data

e.g. horse template has 2 heads!



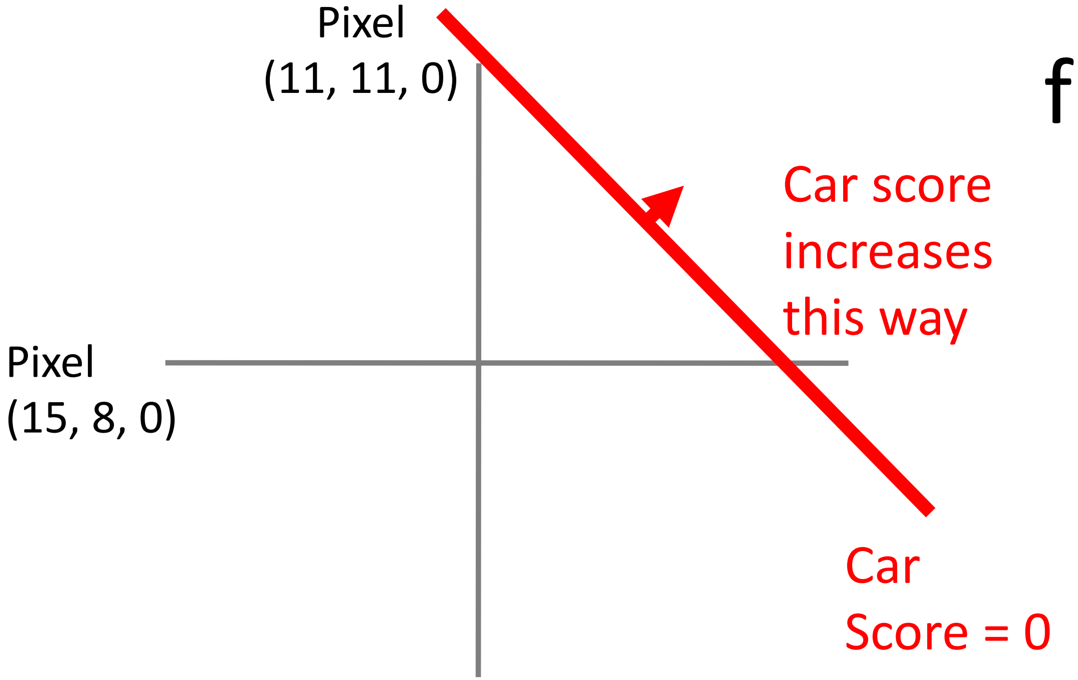

# Interpreting a Linear Classifier: Geometric Viewpoint




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

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

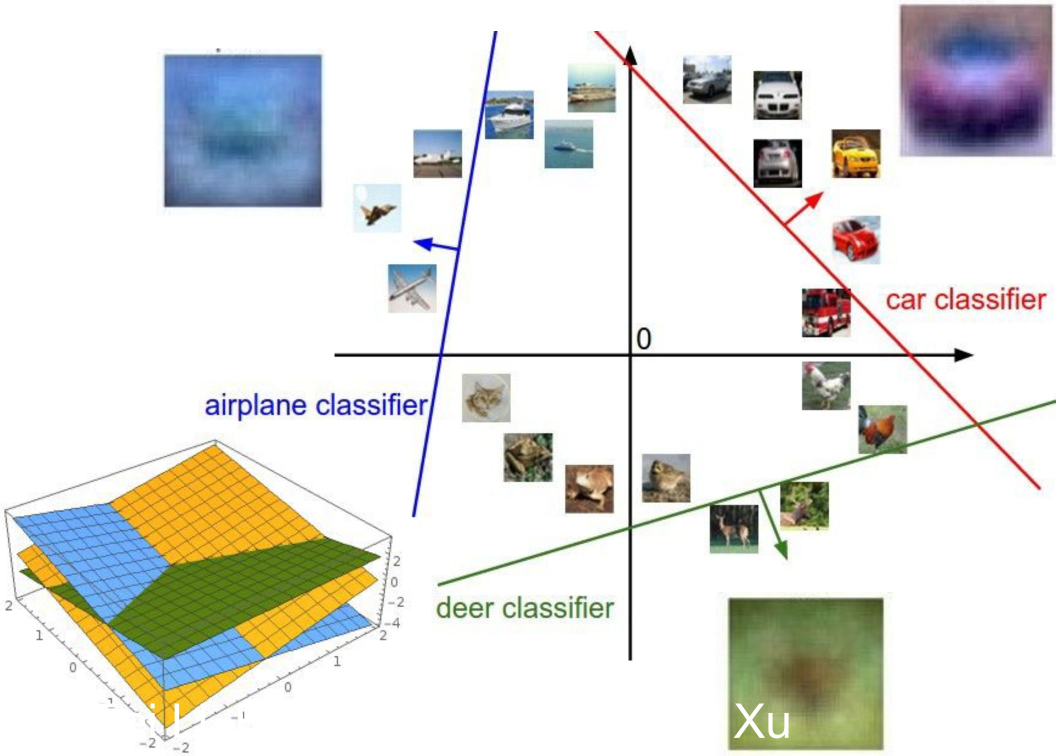
# Interpreting a Linear Classifier: Geometric Viewpoint




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

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

# Interpreting a Linear Classifier: Geometric Viewpoint




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

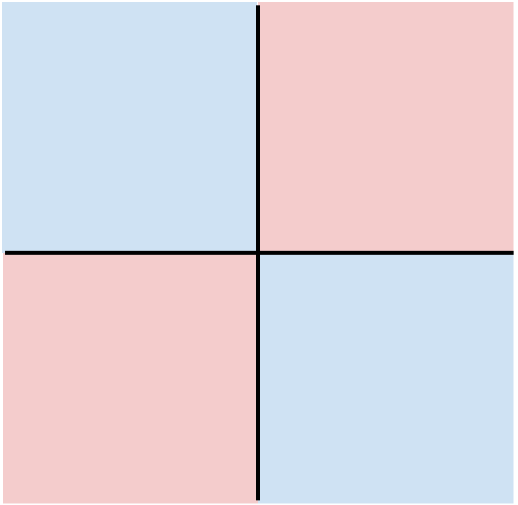
Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

# Hard Cases for a Linear Classifier

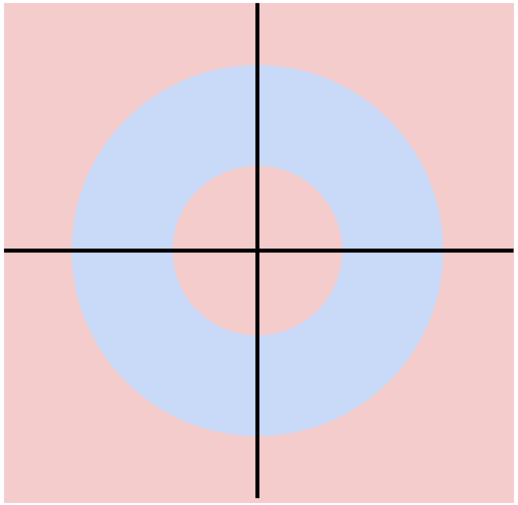
**Class 1**:
First and third quadrants

**Class 2**:
Second and fourth quadrants



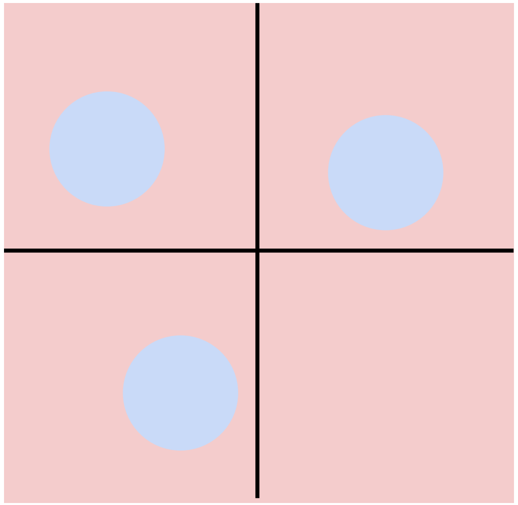
**Class 1**:
1 <= L2 norm <= 2

**Class 2**:
Everything else



**Class 1**:
Three modes

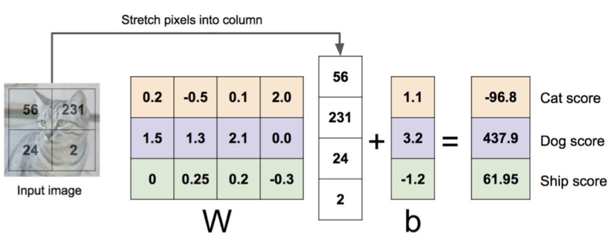
**Class 2**:
Everything else



# Linear Classifier: Three Viewpoints

## Algebraic Viewpoint

$f(x,W) = Wx$

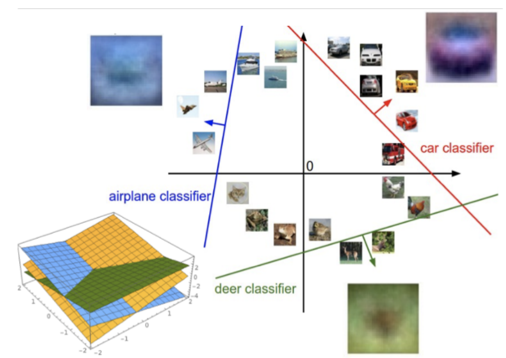



## Visual Viewpoint

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?

# 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

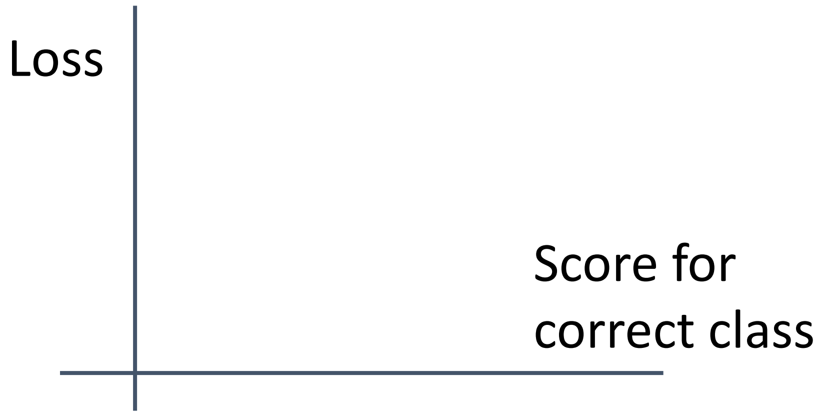
$$L_i(f(x_i, W), y_i)$$

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

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

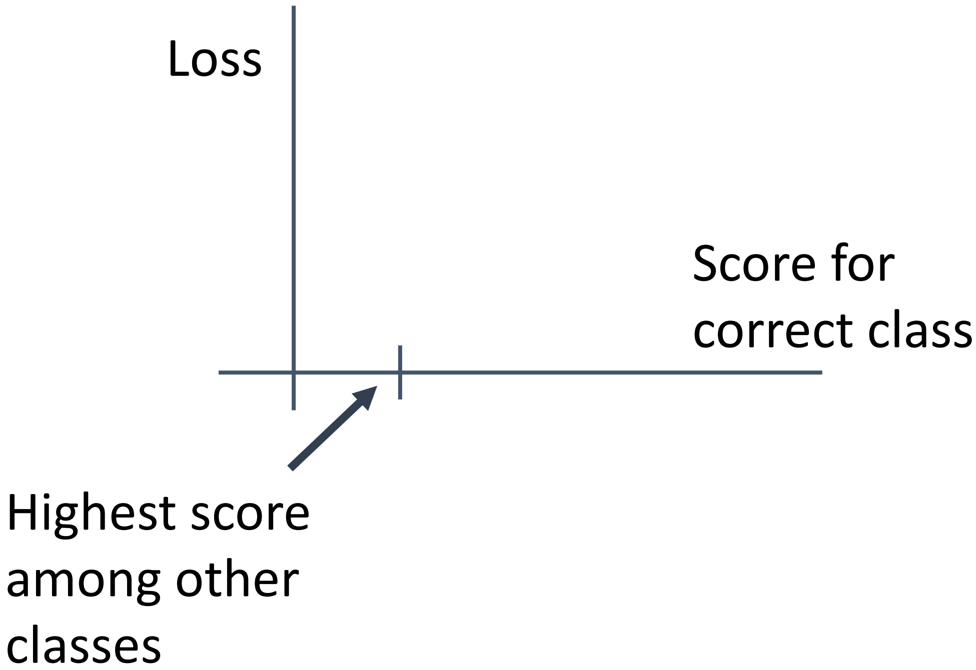
# Multiclass SVM Loss

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



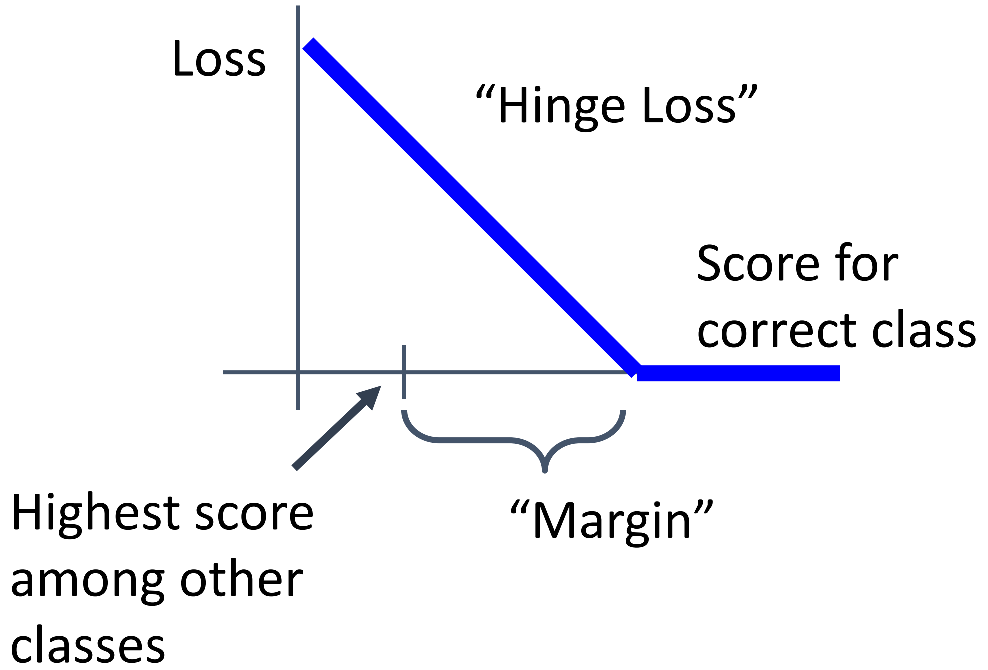
# Multiclass SVM Loss

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



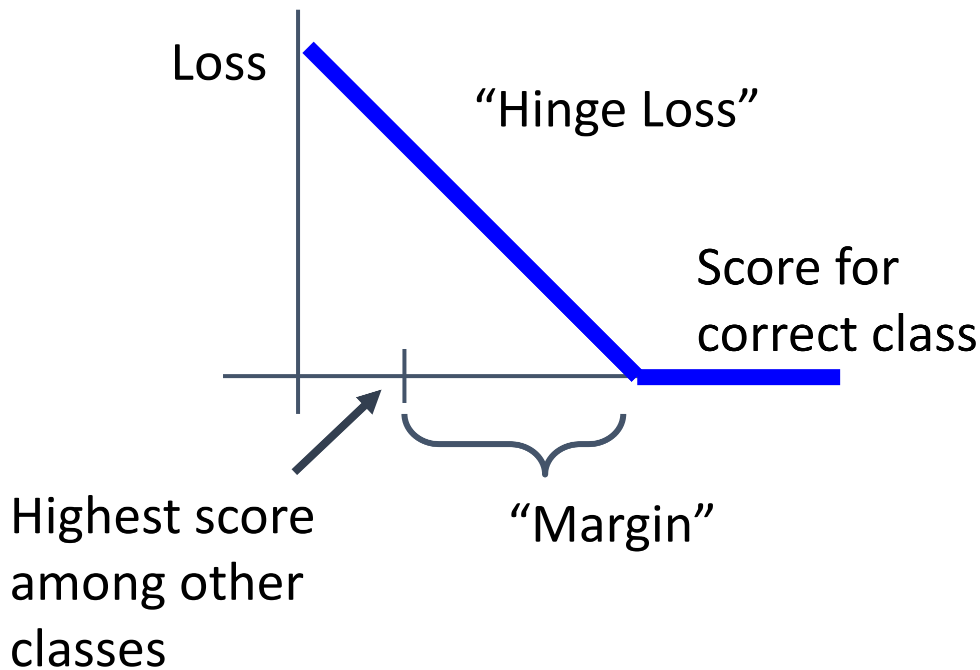
# Multiclass SVM Loss

”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"




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

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

**Data loss**: Model predictions should match training data

# Regularization: Beyond Training Error

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

**Data loss**: Model predictions should match training data

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

# Regularization: Beyond Training Error

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

$\lambda$ = 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

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

$\lambda$ = regularization strength (hyperparameter)

**Data loss**: Model predictions should match training data

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

**Simple examples**

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

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

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

$\lambda$ = regularization strength (hyperparameter)

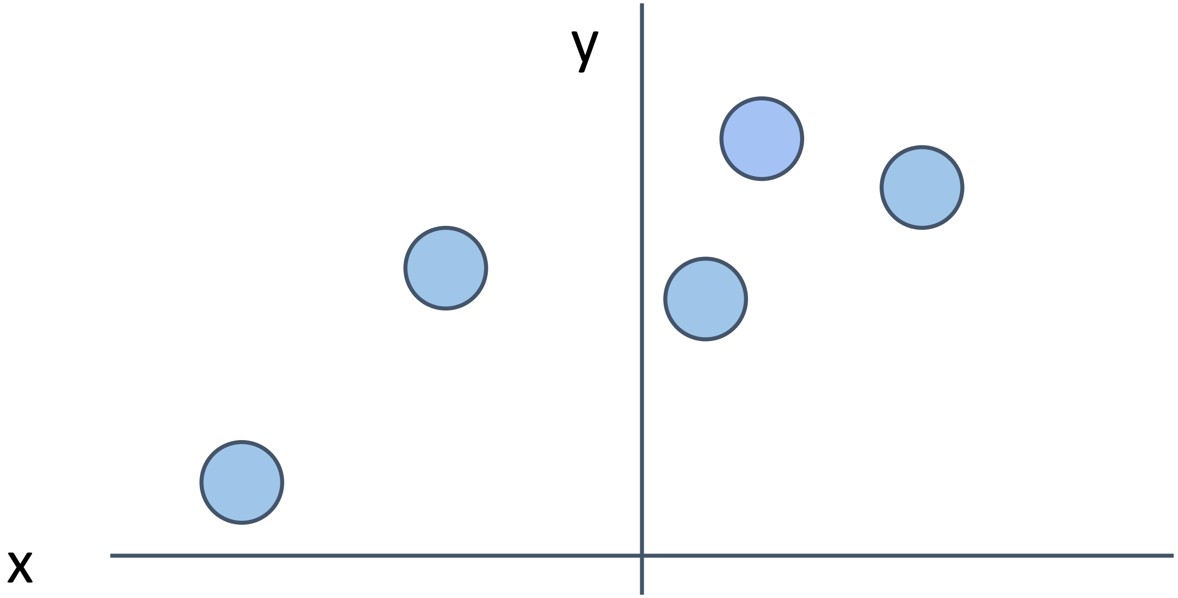
**Data loss**: Model predictions should match training data

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

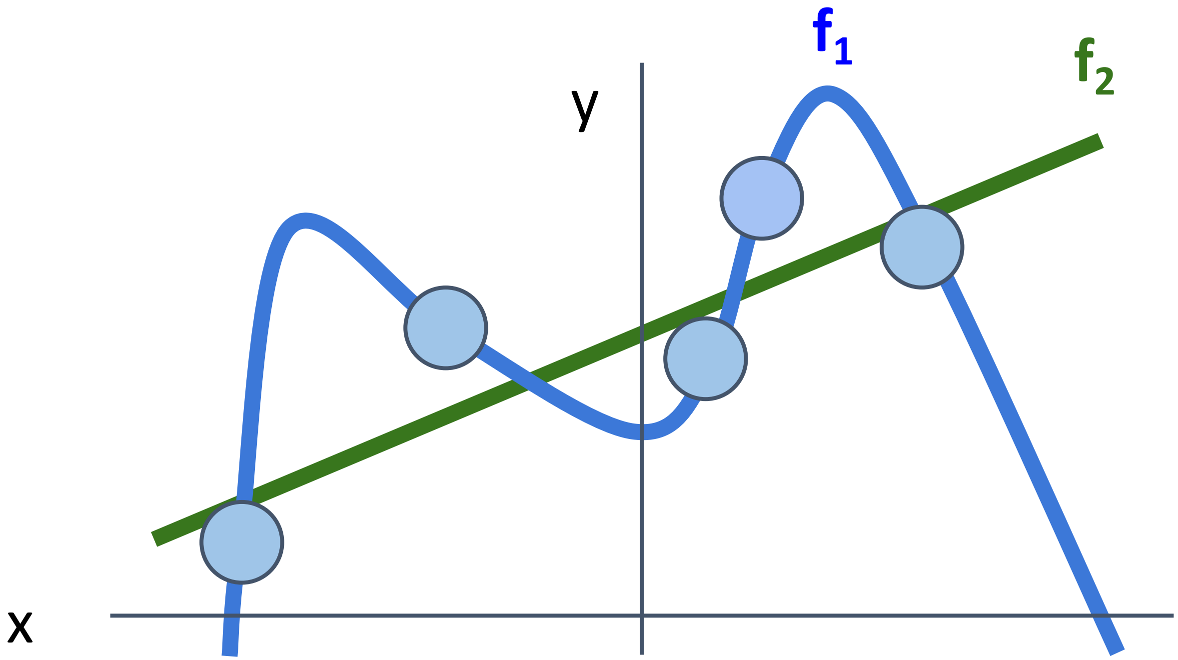
**Purpose of Regularization:**

- 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

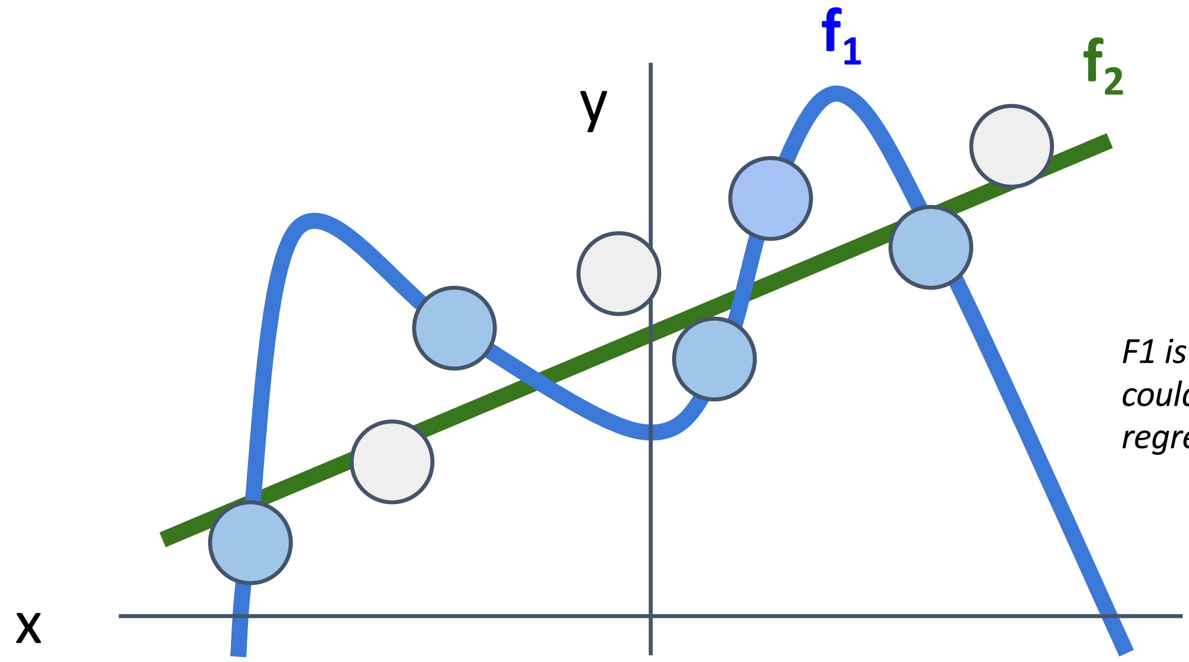


# Regularization: Prefer Simpler Models




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

# Regularization: Prefer Simpler Models

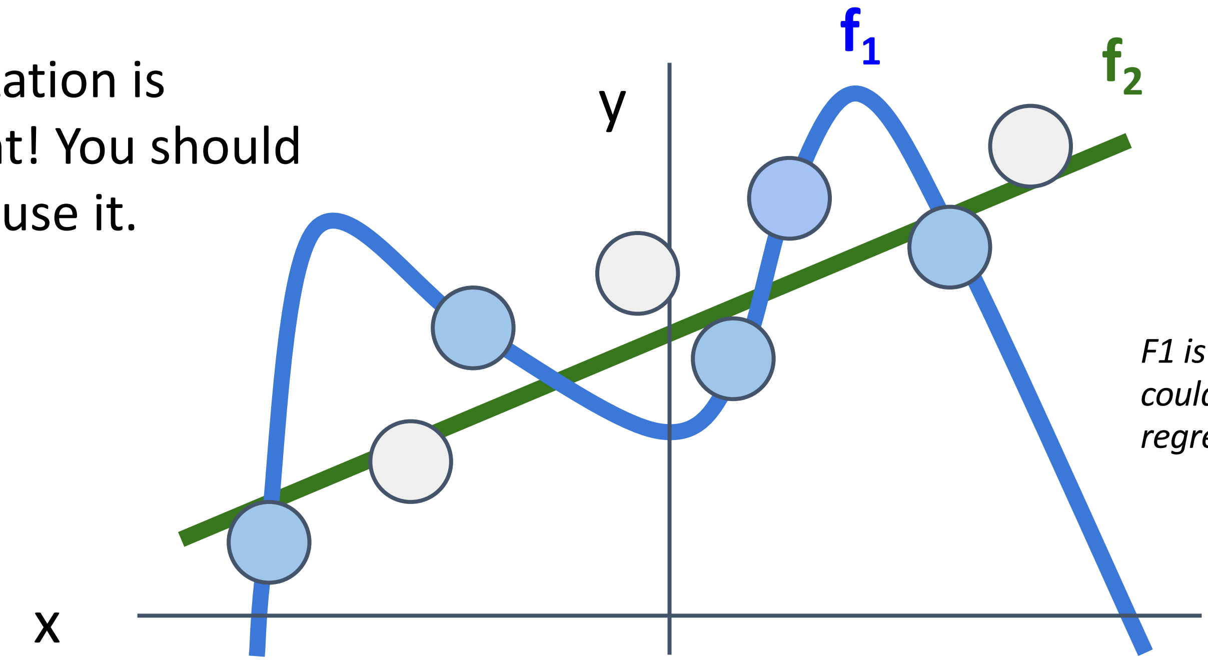



*F1 is not a linear model; could be polynomial regression, etc*

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.




*F1 is not a linear model; could be polynomial regression, etc*

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**

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

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

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**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

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**

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

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

Probabilities must be >= 0

| | | | |
|---|---|---|---|
| cat | **3.2** | | **24.5** |
| car | 5.1 | exp $\rightarrow$ | 164.0 |
| frog | -1.7 | | 0.18 |

Unnormalized log-probabilities / logits

unnormalized probabilities

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

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

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

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | $\xrightarrow{\text{exp}}$ | unnormalized probabilities | $\xrightarrow{\text{normalize}}$ | probabilities |
|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

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

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

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

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities |
|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

$L_i = -\log(0.13)$

$= \mathbf{2.04}$

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

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

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

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

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities |
|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |

Probabilities must be >= 0

Probabilities must sum to 1

$L_i = -\log(0.13)$
$= \mathbf{2.04}$

**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**

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

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

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

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities | | | | Correct probs |
|---|---|---|---|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** | → | Compare | ← | **1.00** |
| car | 5.1 | | 164.0 | | 0.87 | | | | 0.00 |
| frog | -1.7 | | 0.18 | | 0.00 | | | | 0.00 |

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

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

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

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

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | | unnormalized probabilities | | probabilities | | Correct probs |
|---|---|---|---|---|---|---|---|
| cat | **3.2** | exp → | **24.5** | normalize → | **0.13** | → Compare ← | **1.00** |
| car | 5.1 | | 164.0 | | 0.87 | | 0.00 |
| frog | -1.7 | | 0.18 | | 0.00 | | 0.00 |

*Kullback–Leibler divergence*

$$D_{KL}(P \| Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

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

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

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

Probabilities must be >= 0

Probabilities must sum to 1

| | Unnormalized log-probabilities / logits | $\xrightarrow{\text{exp}}$ | unnormalized probabilities | $\xrightarrow{\text{normalize}}$ | probabilities | | Correct probs |
|---|---|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** | → Compare ← | **1.00** |
| car | 5.1 | | 164.0 | | 0.87 | | 0.00 |
| frog | -1.7 | | 0.18 | | 0.00 | | 0.00 |

*Cross Entropy*

$$H(P, Q) = H(p) + D_{KL}(P \| Q)$$

# Cross-Entropy Loss (Multinomial Logistic Regression)

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

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

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

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**

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

$$\boxed{s = f(x_i; W)}$$

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

Maximize probability of correct class

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

Putting it all together:

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

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

# Cross-Entropy Loss (Multinomial Logistic Regression)

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

$$\boxed{s = f(x_i; W)}$$

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

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Maximize probability of correct class

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

Putting it all together:

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

**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**

$$\boxed{s = f(x_i; W)}$$

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

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Maximize probability of correct class

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

Putting it all together:

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

**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**

$$\boxed{s = f(x_i; W)}$$

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

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

Maximize probability of correct class

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

Putting it all together:

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

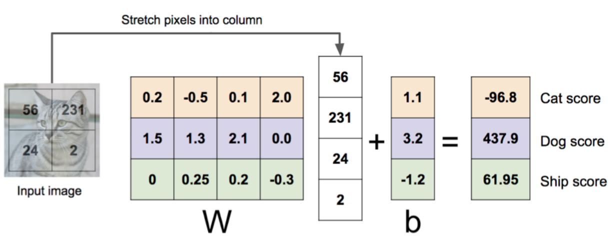
**Q:** If all scores are small random values, what is the loss?

**A**: -log(1/C)
log(10) ≈ 2.3

# Recap: Three ways to think about linear classifiers

## Algebraic Viewpoint

$f(x,W) = Wx$

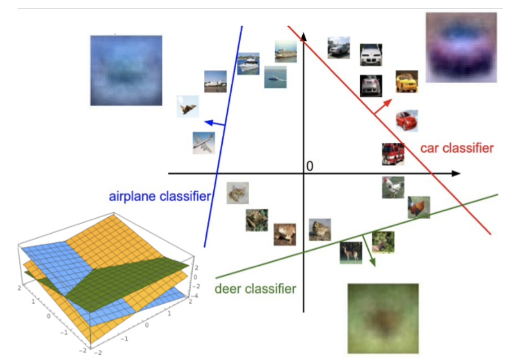



## Visual Viewpoint

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**:

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

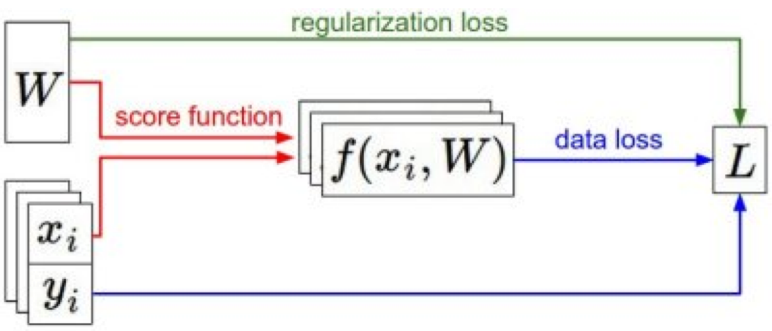
Linear classifier

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

SVM

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

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

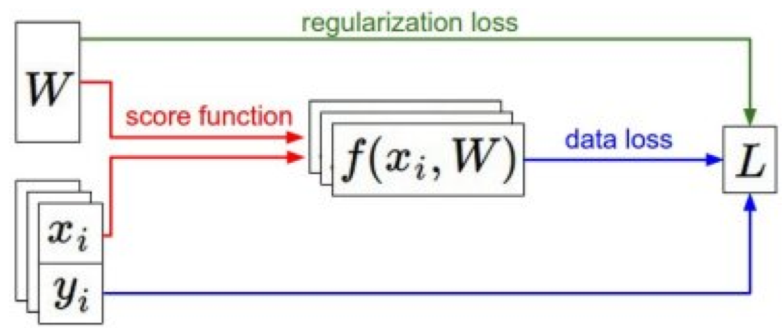
$$s = f(x; W) = Wx$$

Linear classifier

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

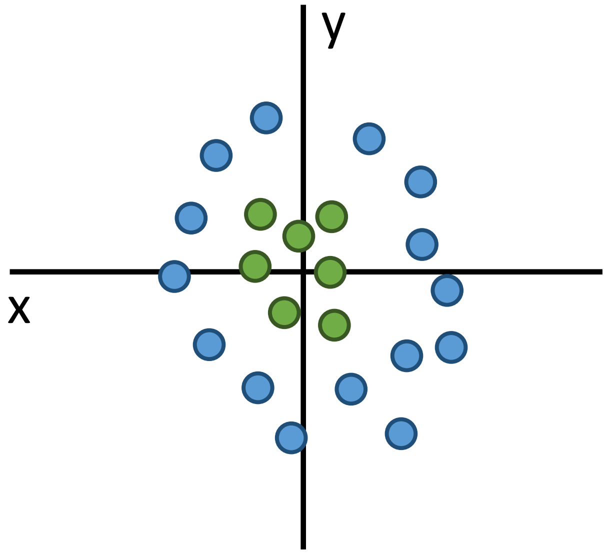
SVM
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



# Problem: Linear Classifiers aren’t that powerful

Geometric Viewpoint

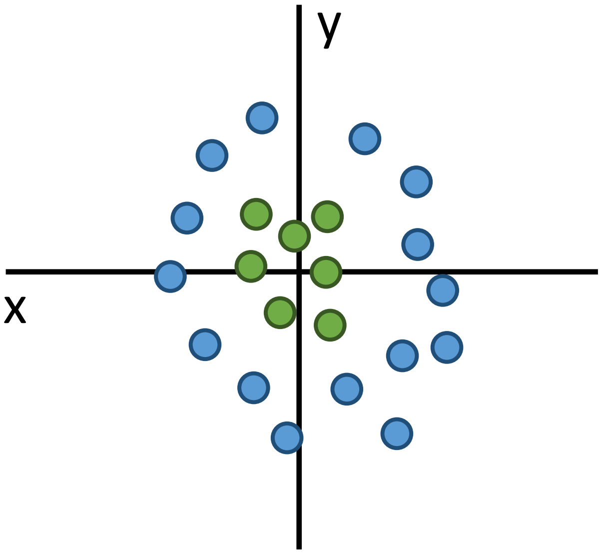



Visual Viewpoint

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

# One solution: **Feature Transforms**

Original space

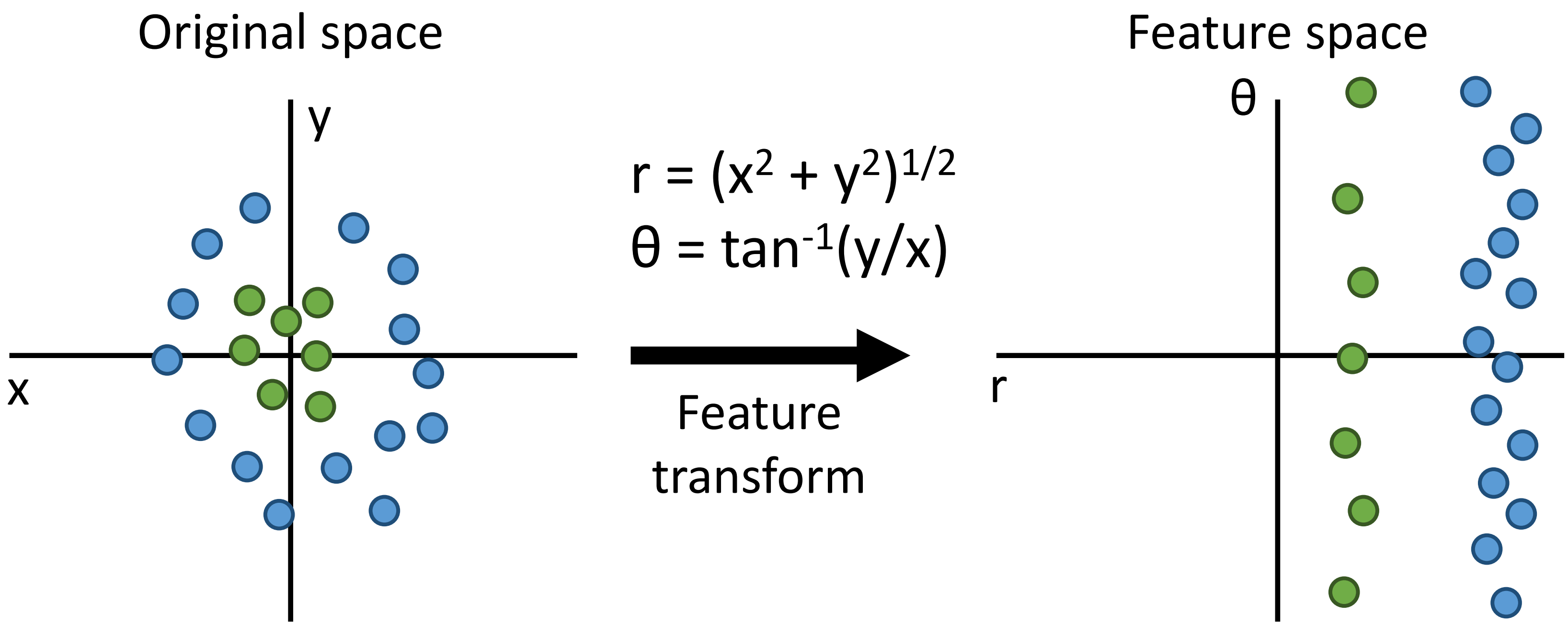



$r = (x^2 + y^2)^{1/2}$

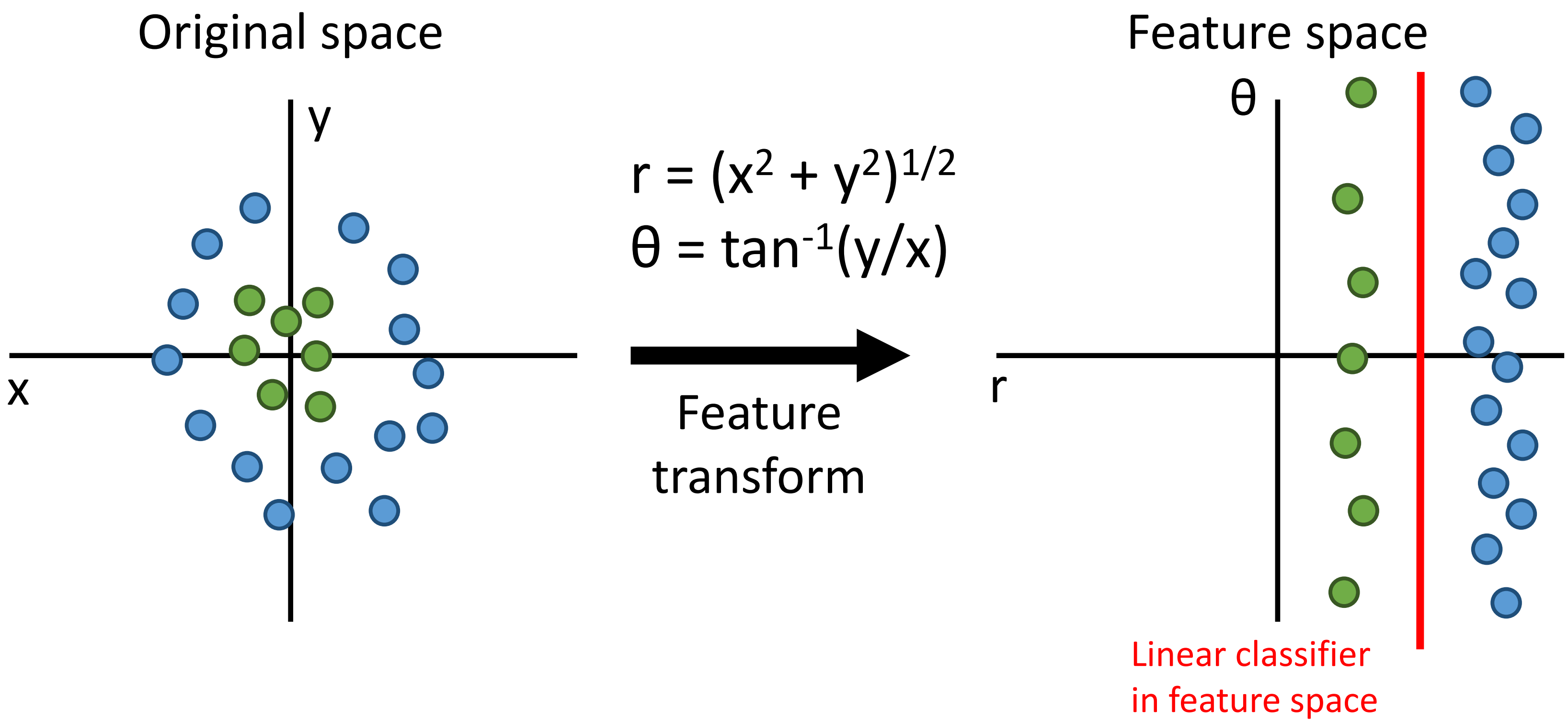
$\theta = \tan^{-1}(y/x)$

Feature transform

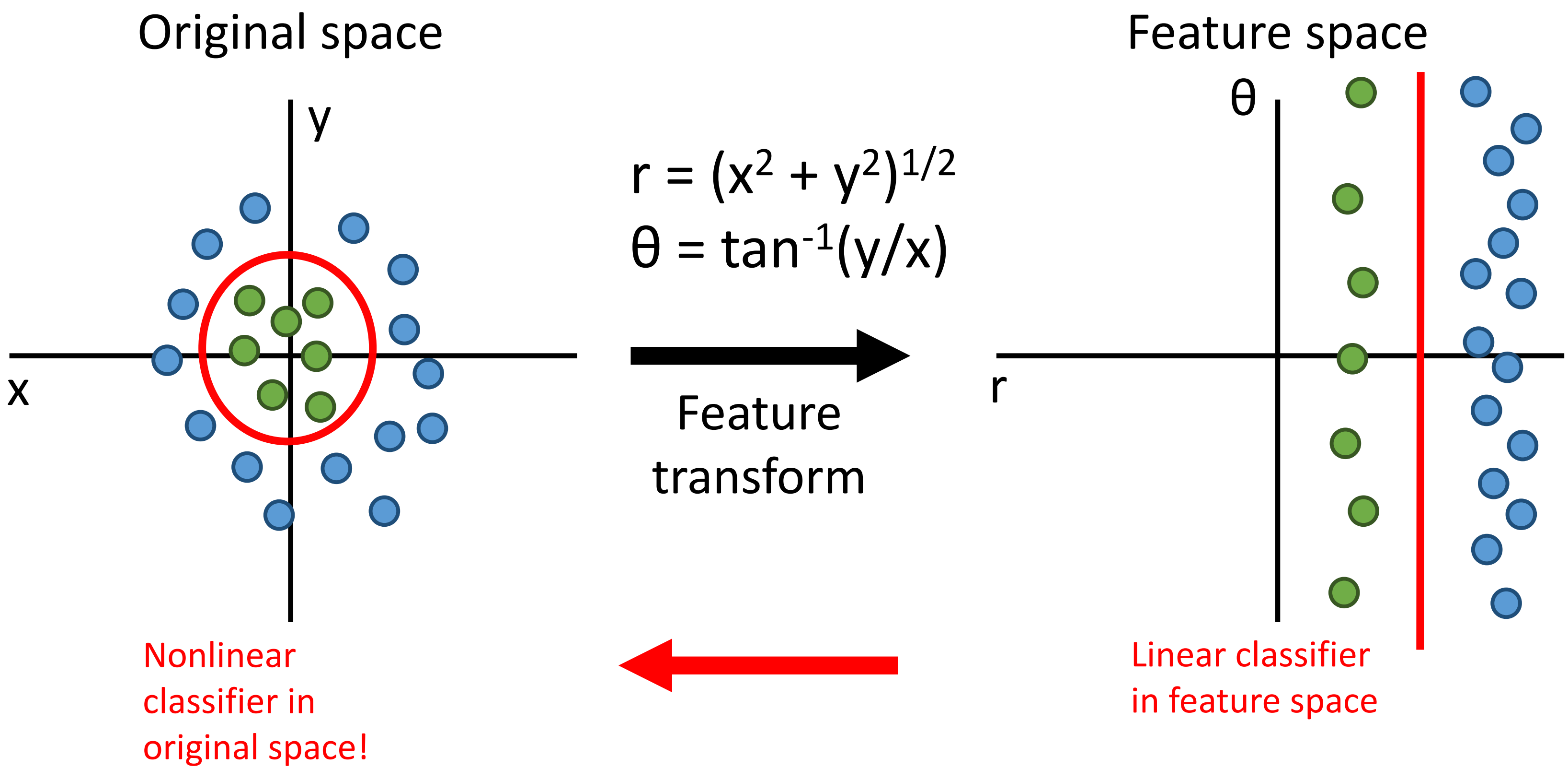
# One solution: **Feature Transforms**



# One solution: **Feature Transforms**



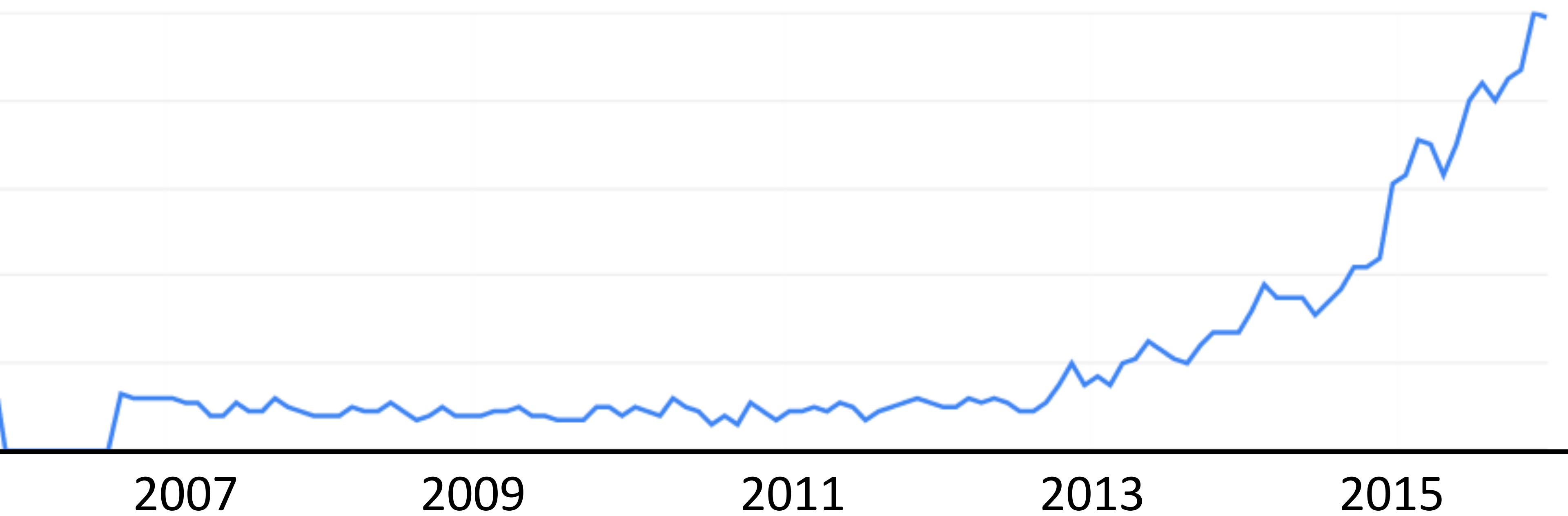
# One solution: **Feature Transforms**



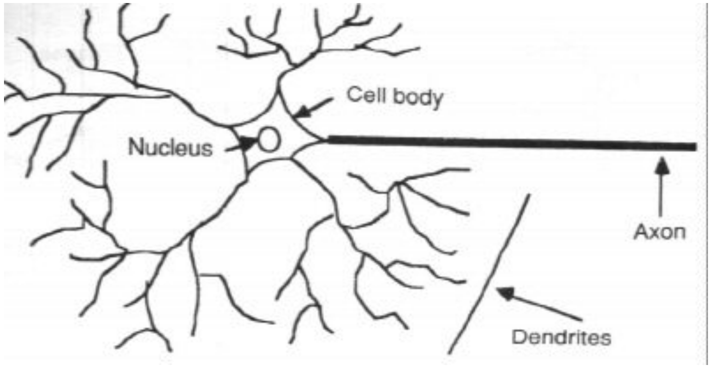

$r = (x^2 + y^2)^{1/2}$
$\theta = \tan^{-1}(y/x)$

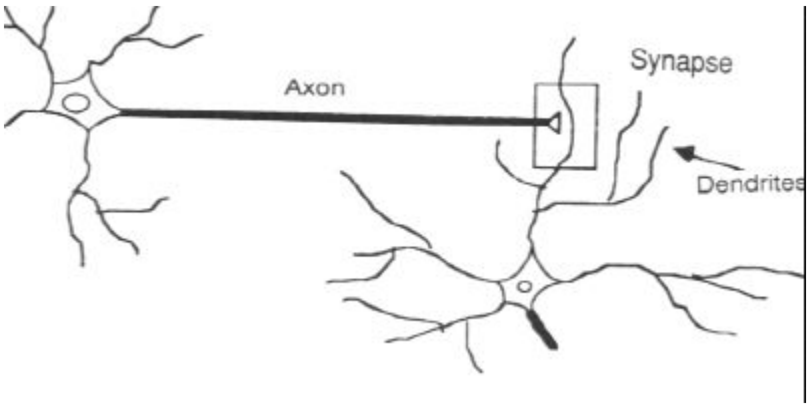
# Deep learning attracts lots of attention.

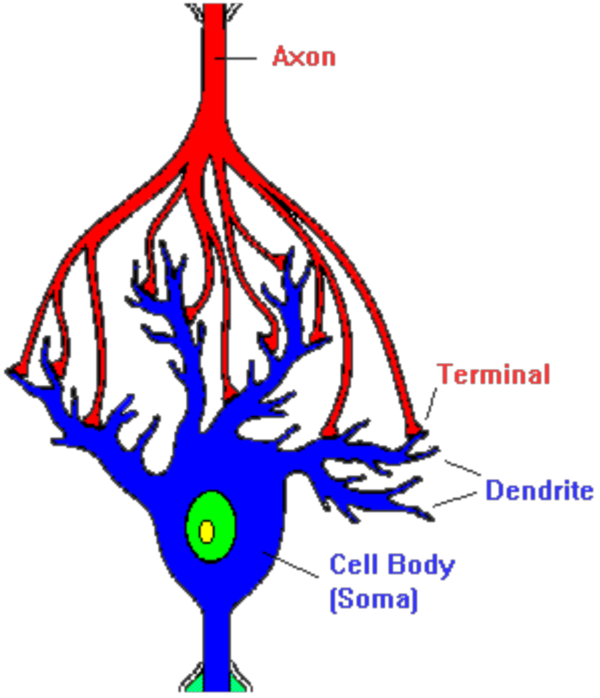
- Google Trends



# How the Human Brain learns

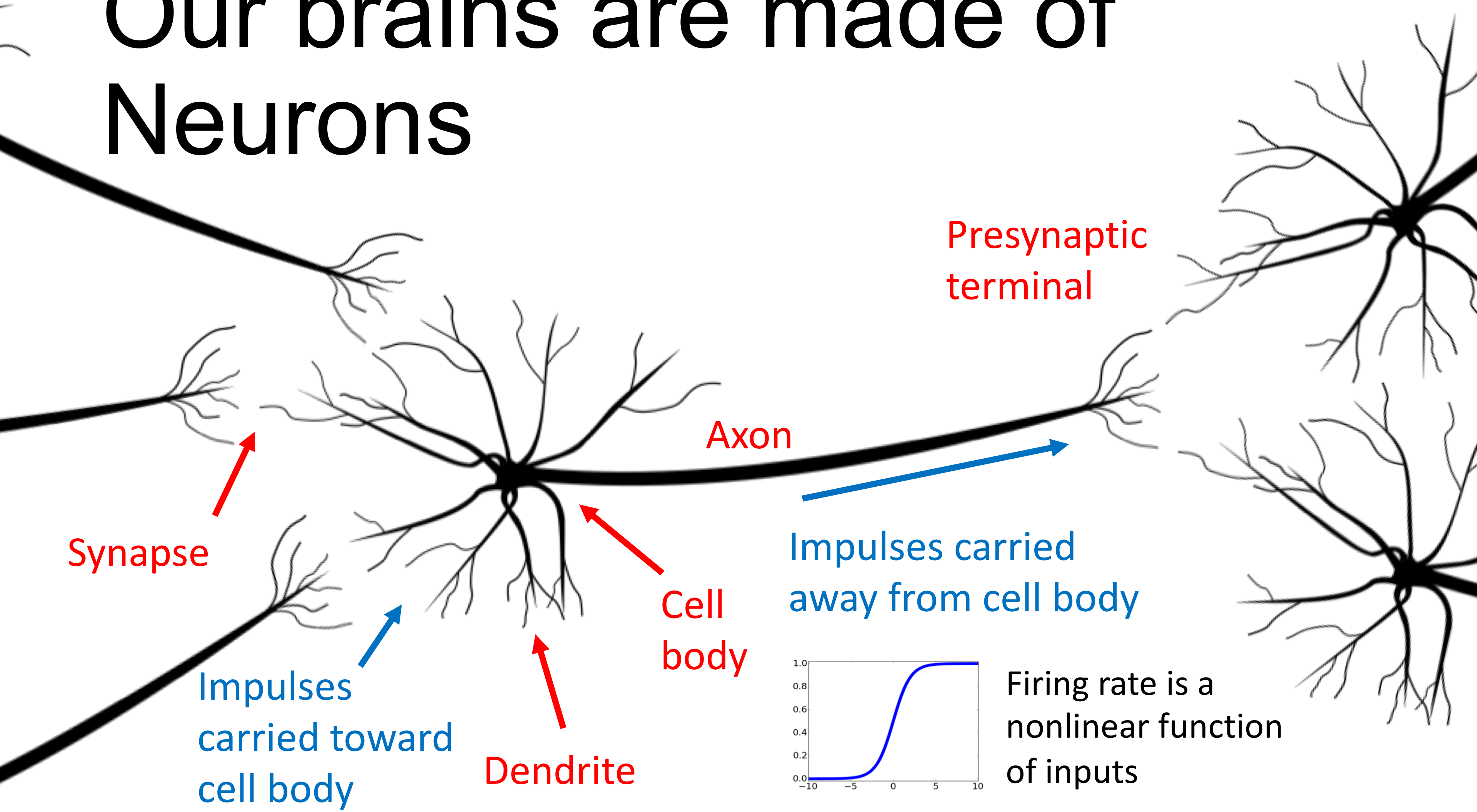


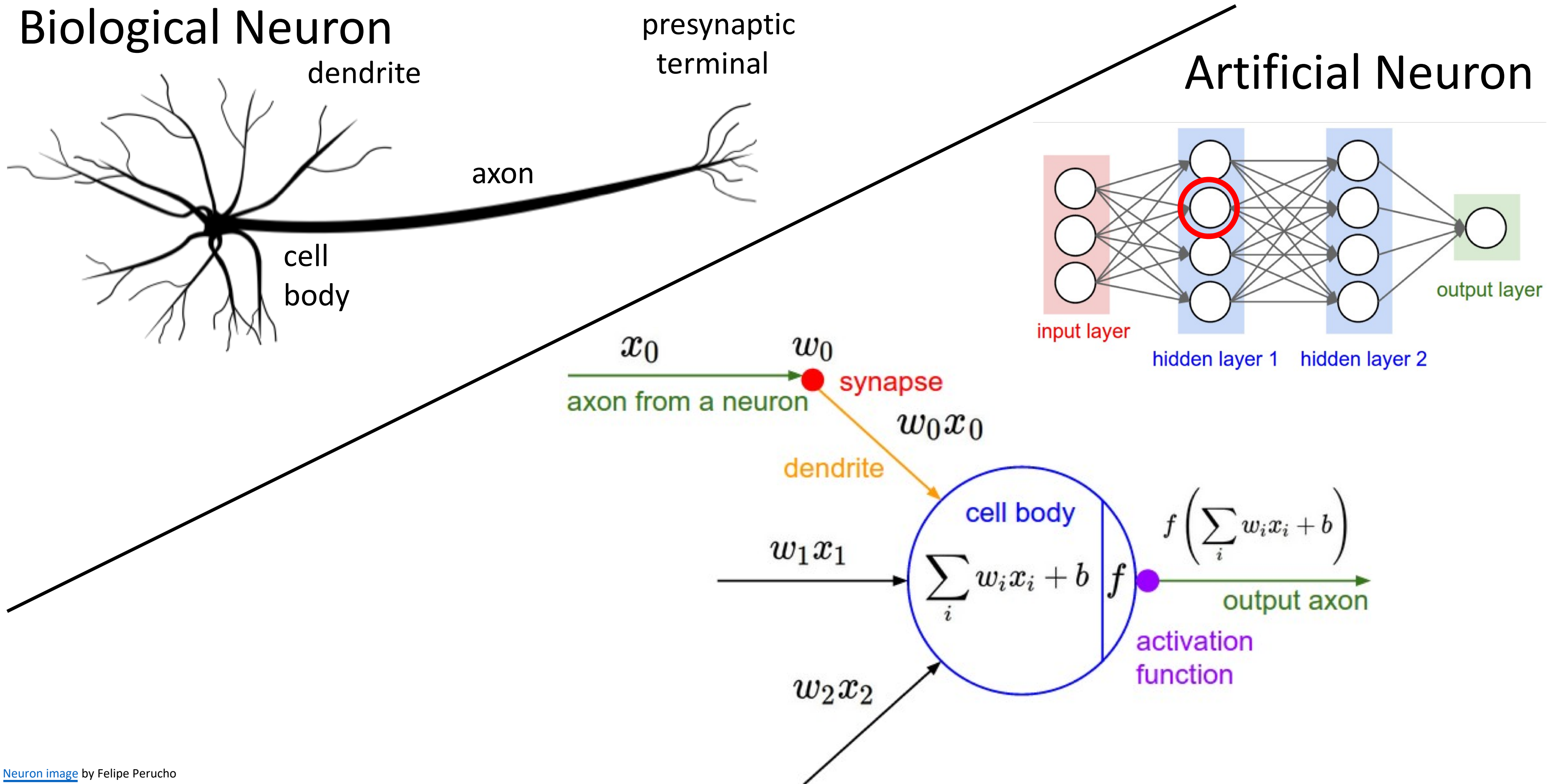




- In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.
- The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches.
- At the end of each branch, a structure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

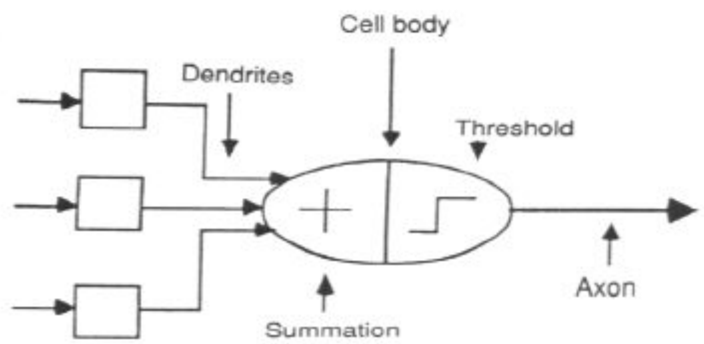




Biological Neuron
dendrite
presynaptic
terminal
axon
cell
body
Artificial Neuron
input layer
hidden layer 1
hidden layer 2
output layer
$x_0$
$w_0$
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$\sum_i w_i x_i + b$
$f$
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
output axon
activation
function
$w_2 x_2$

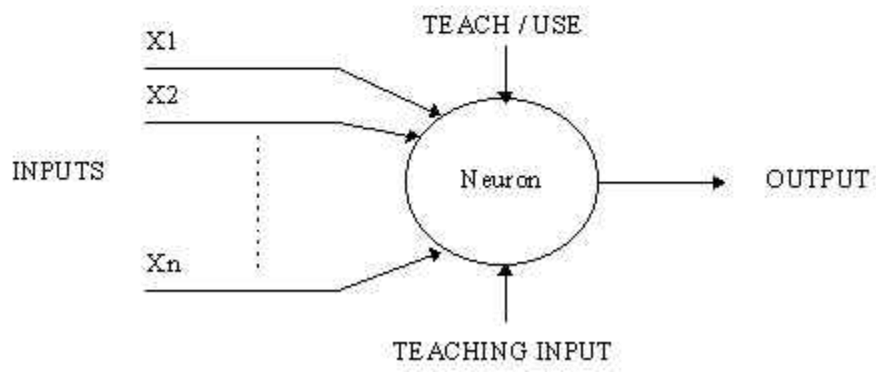
# A Neuron Model

- 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.)

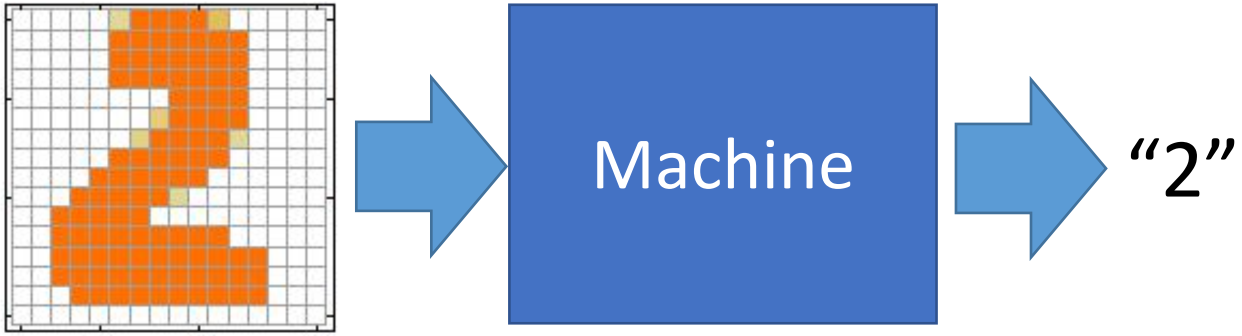
- In the training mode, the neuron can be trained to fire (or not), for particular input patterns.
- In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the 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

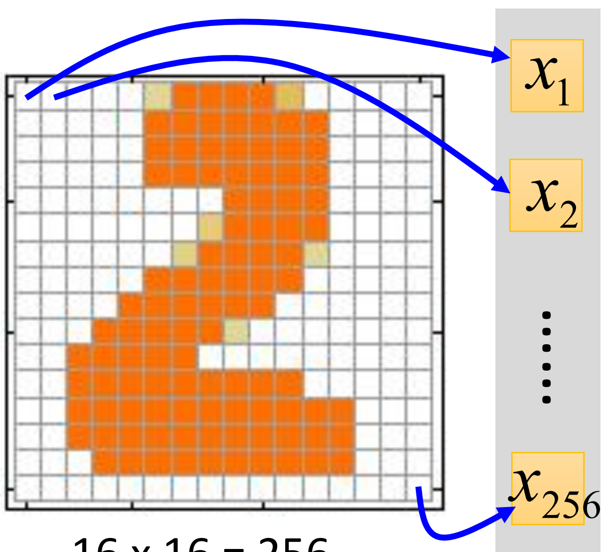
# Example Application

- Handwriting Digit Recognition



# Handwriting Digit Recognition

## Input

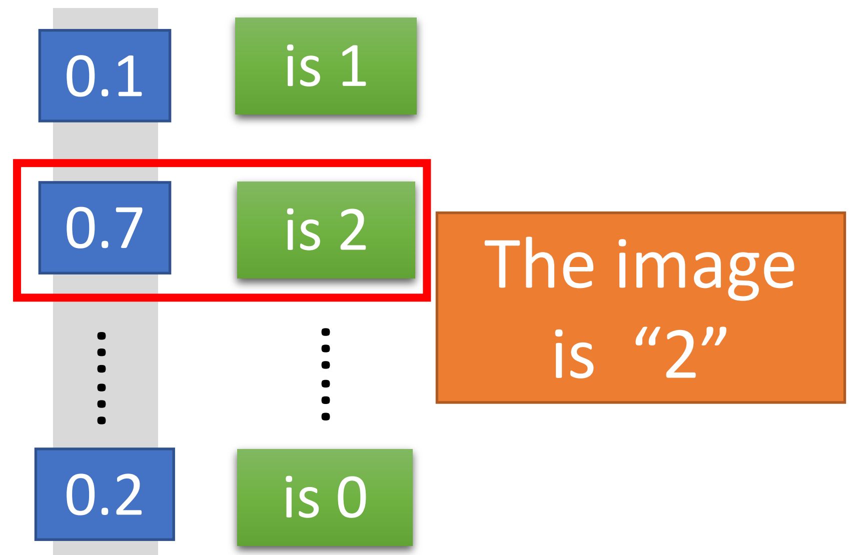



16 x 16 = 256

Ink → 1

No ink → 0

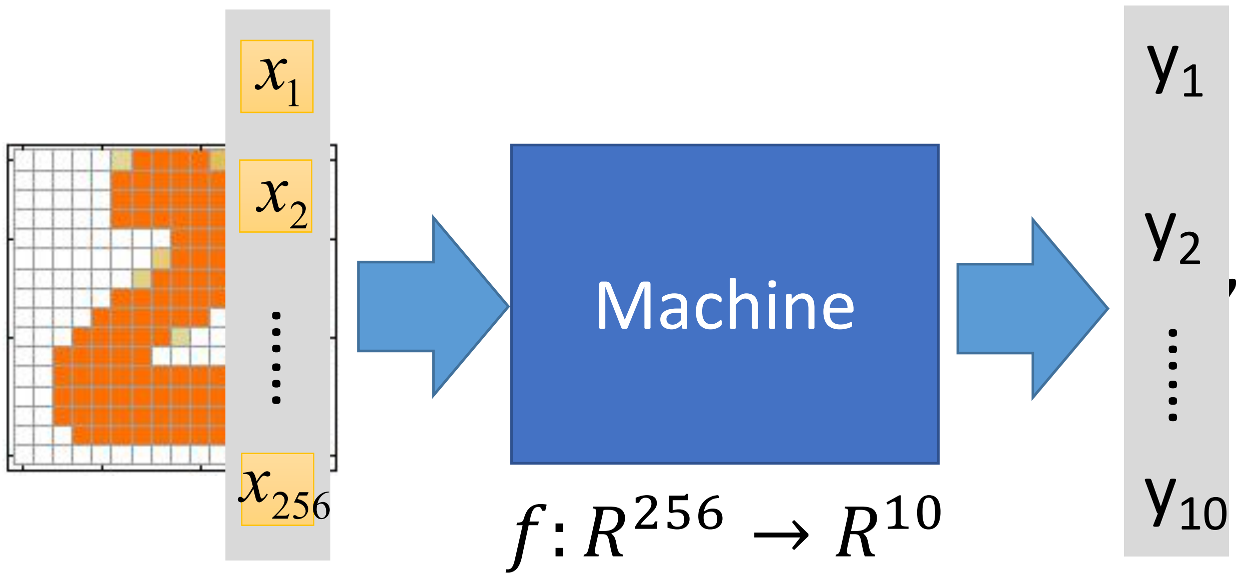
## Output




Each dimension represents the confidence of a digit.

# Example Application

- Handwriting Digit Recognition

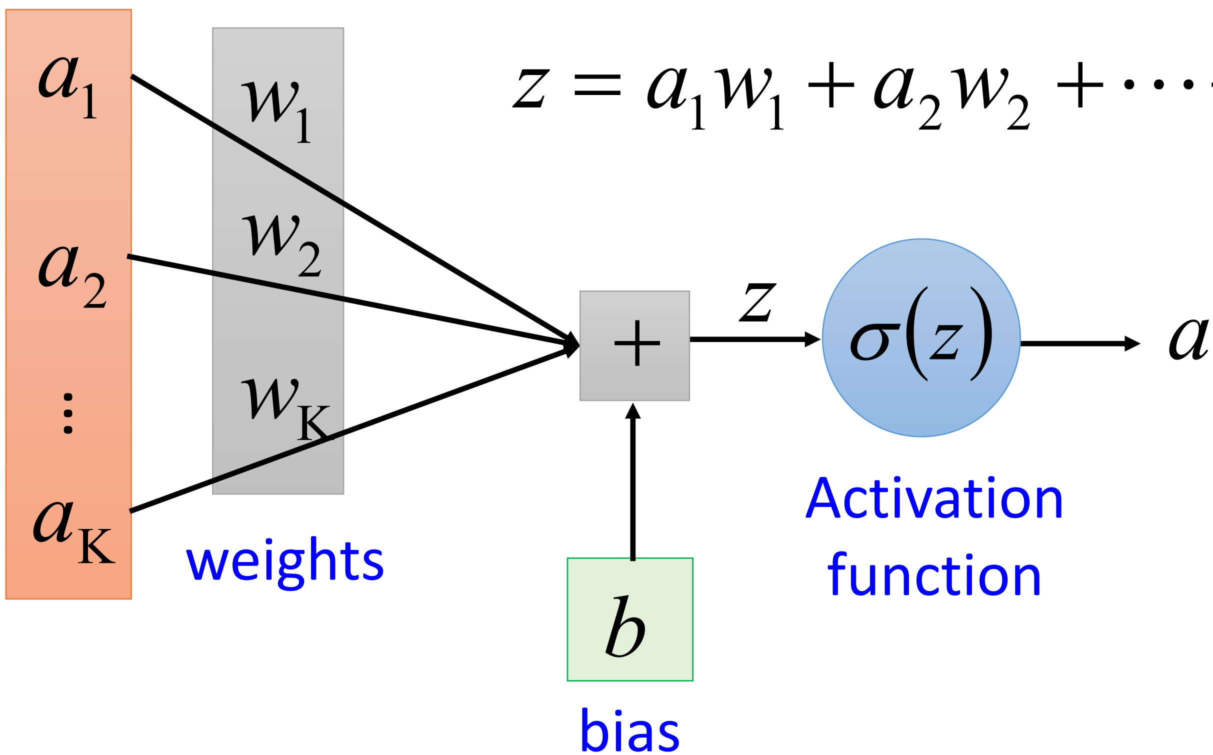



In deep learning, the function $f$ is represented by neural network

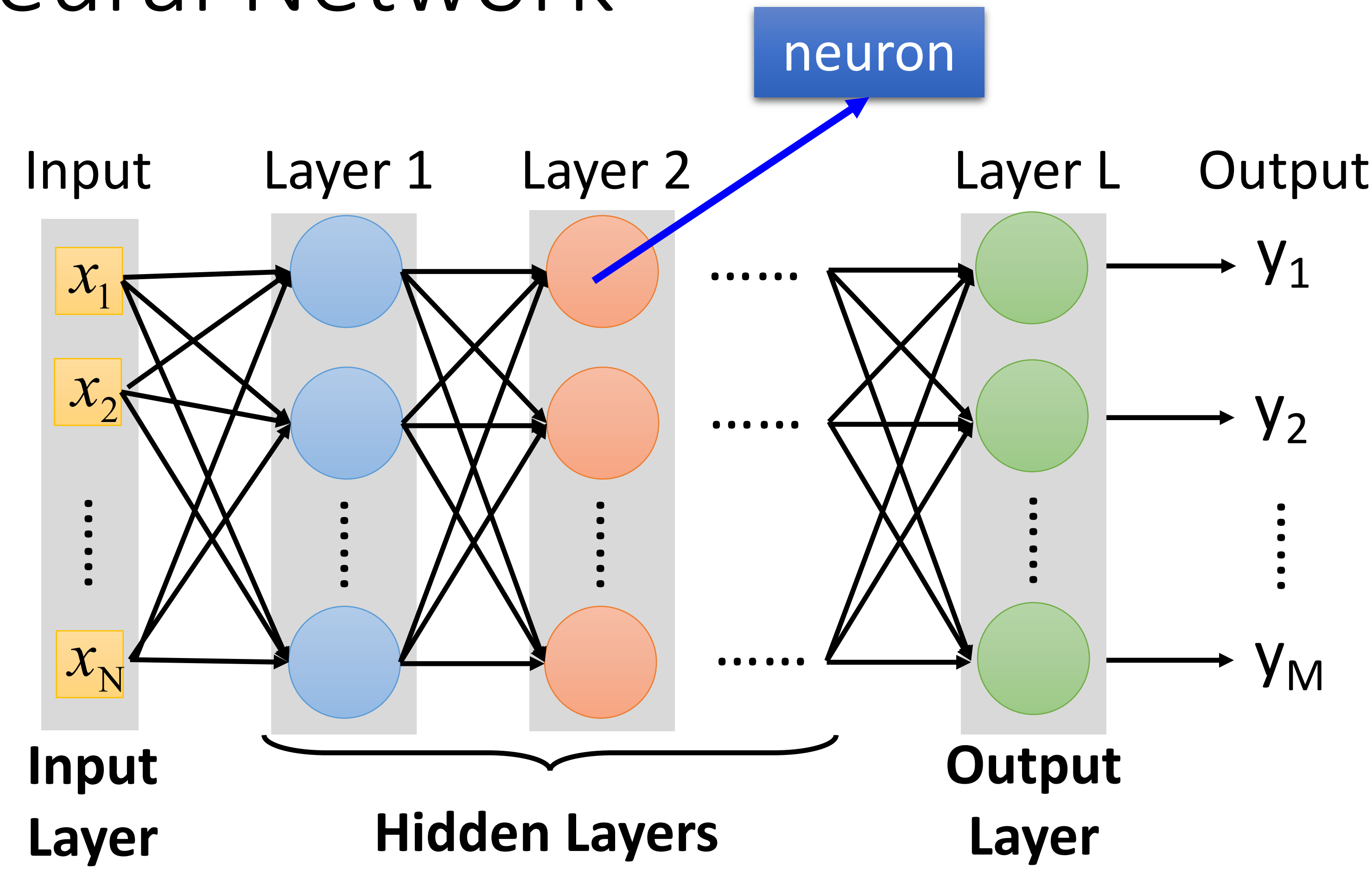
# Element of Neural Network

***Neuron*** $f: R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

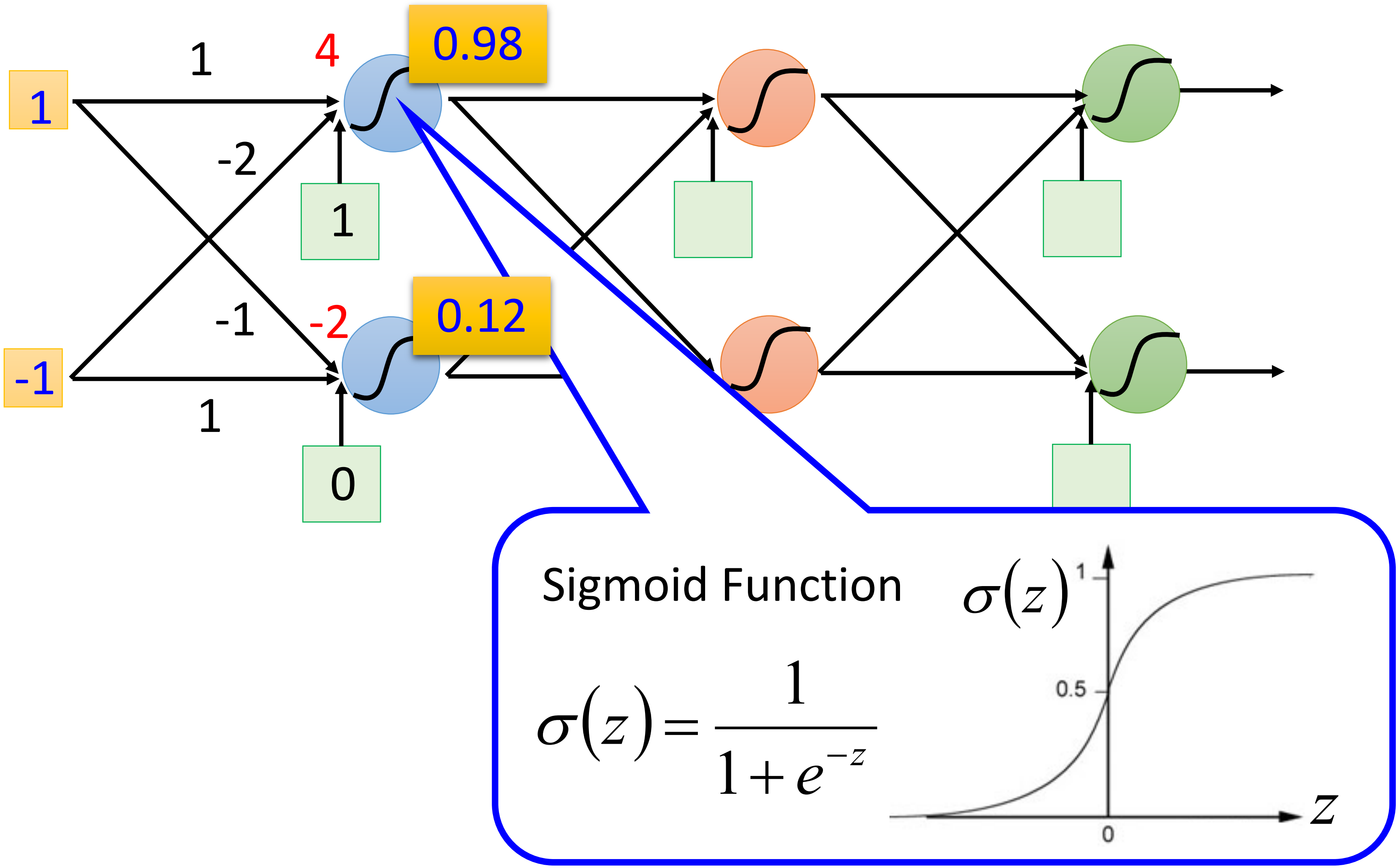


# Neural Network

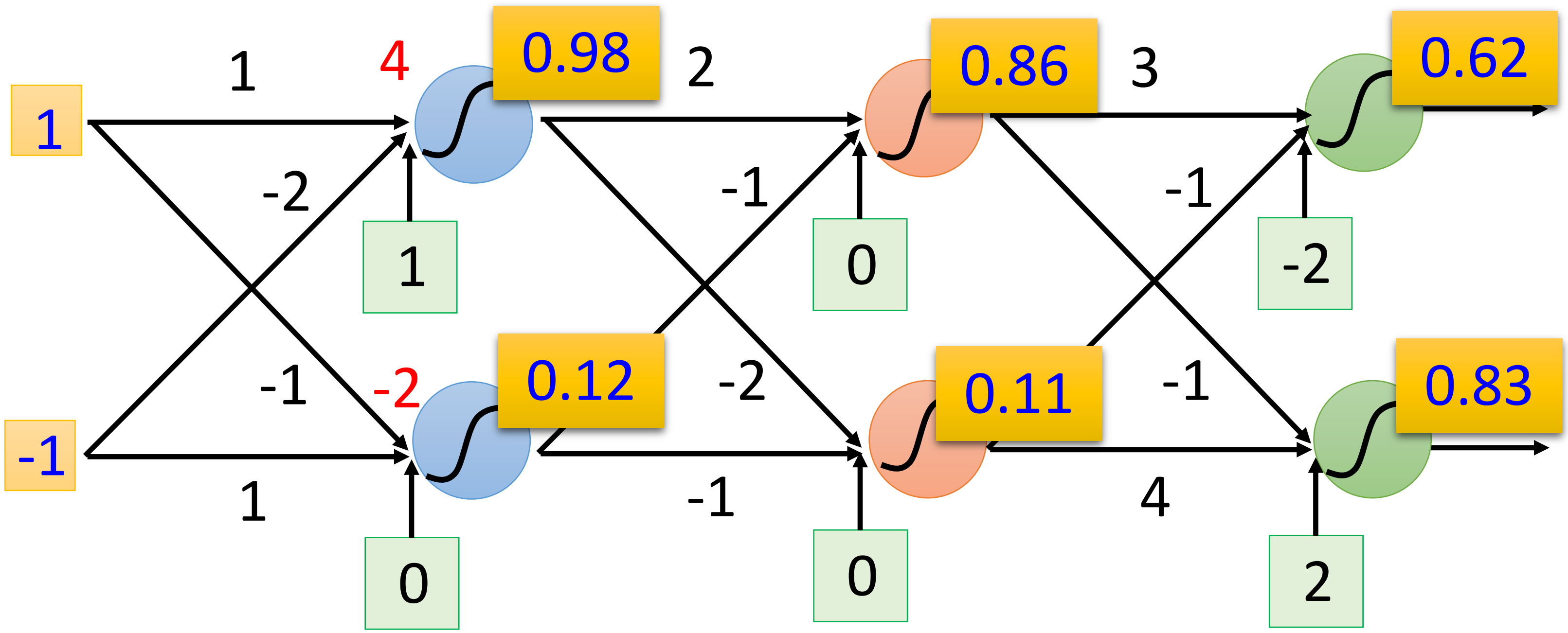



Deep means many hidden layers

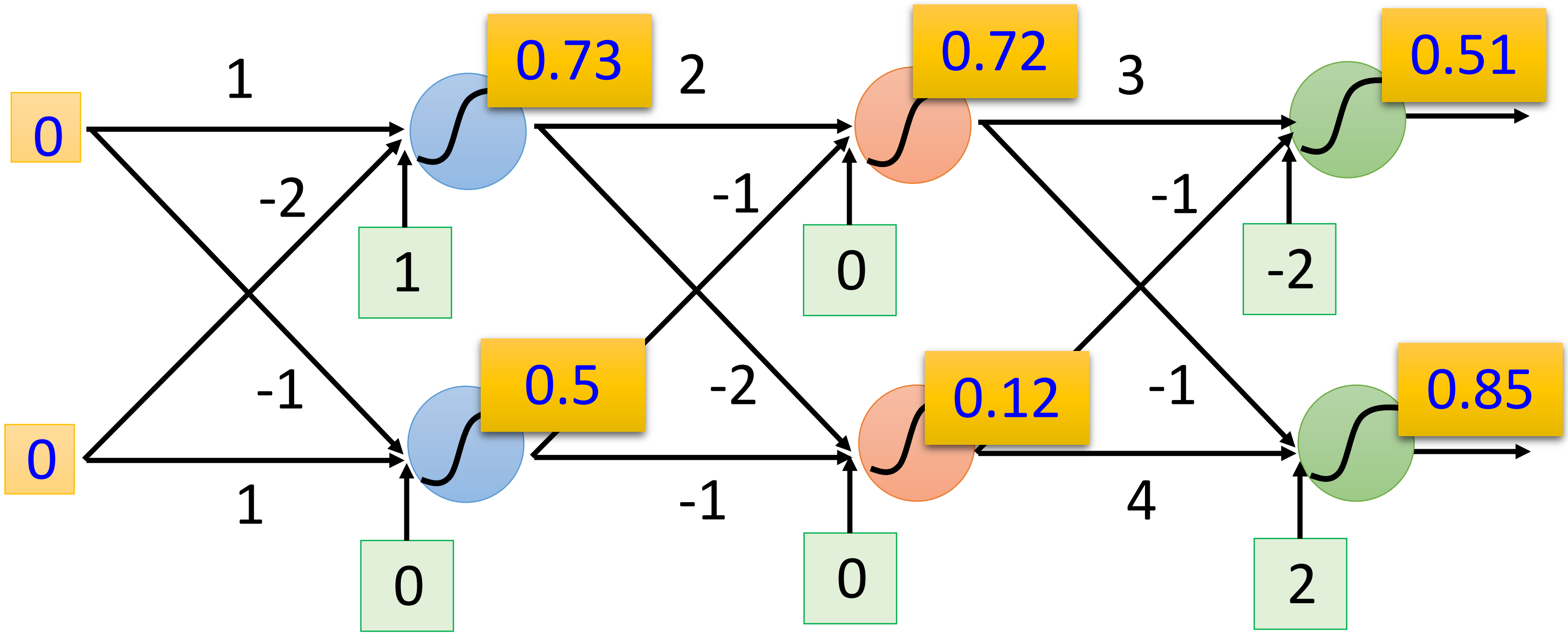
# Example of Neural Network



# Example of Neural Network



# Example of Neural Network

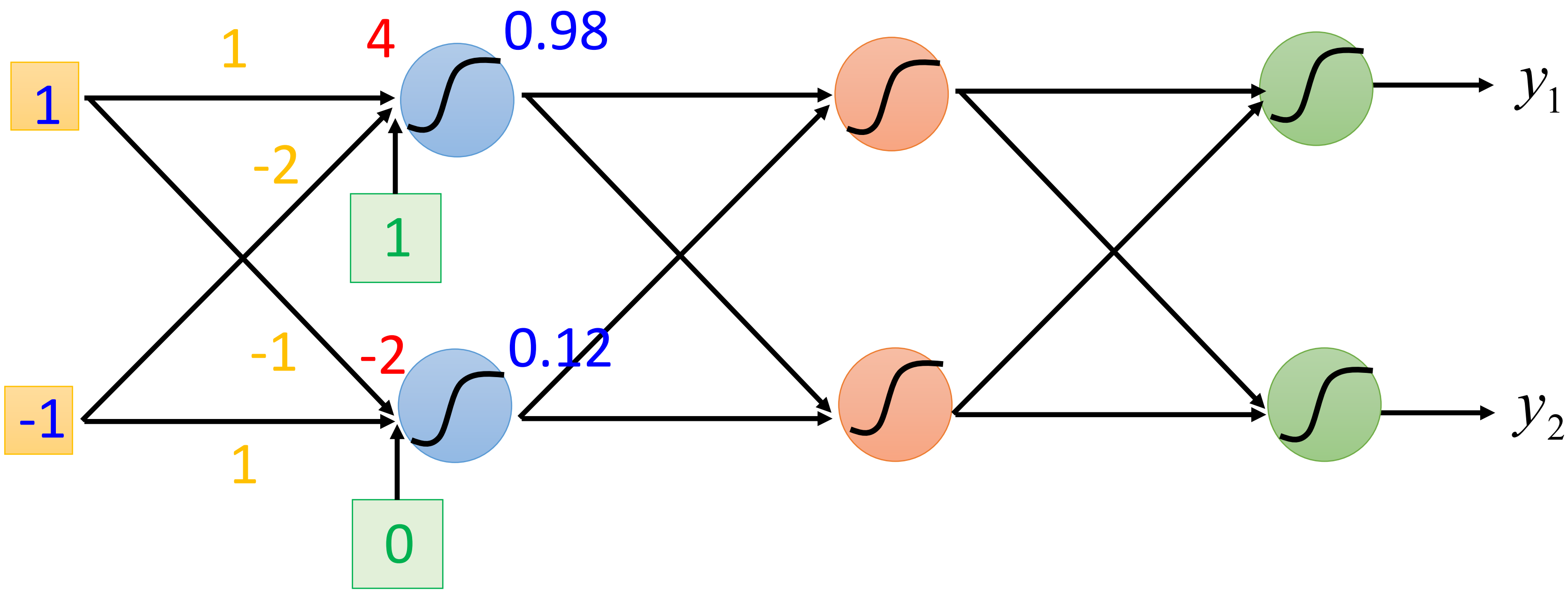



$f: R^2 \to R^2$

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

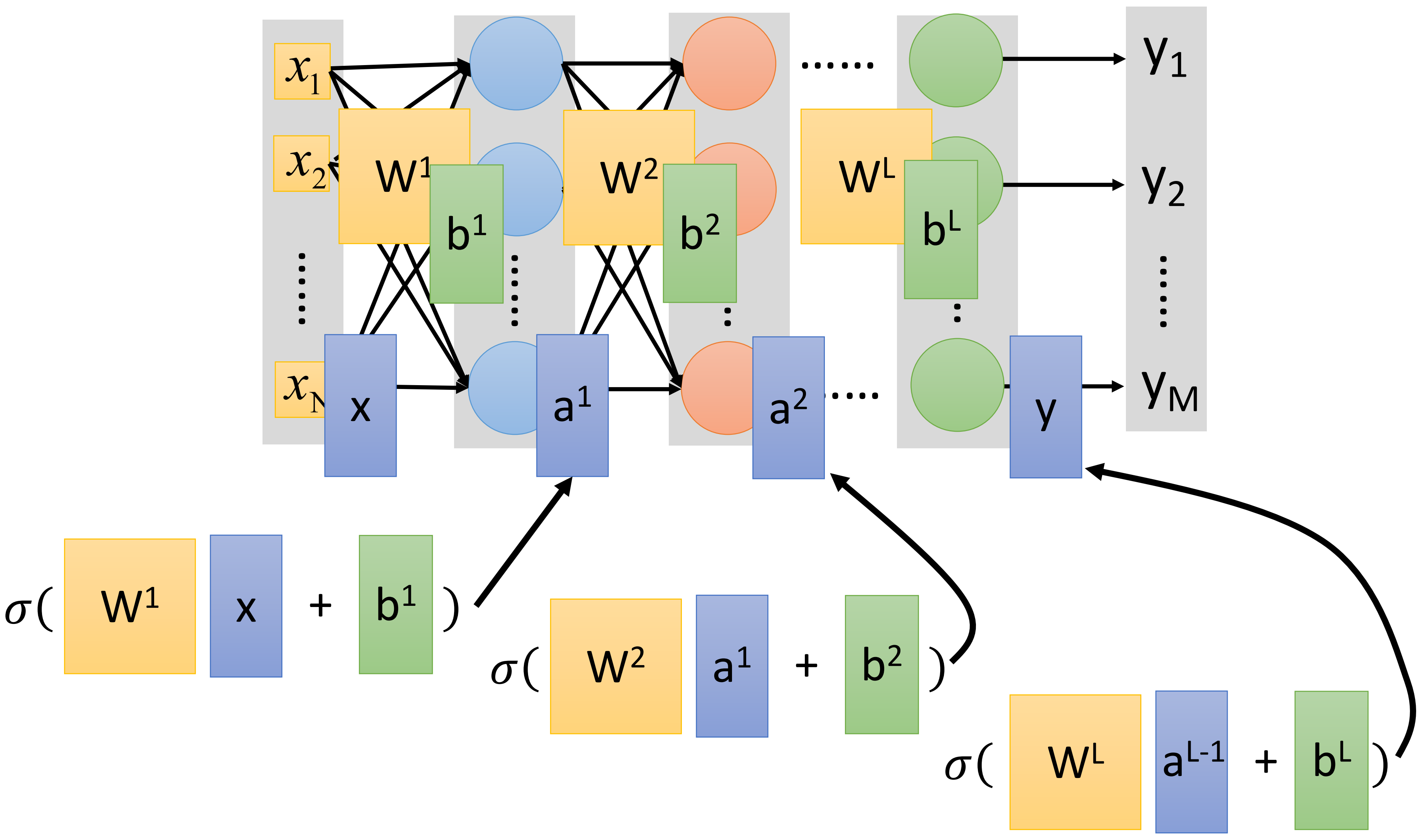
Different parameters define different function

# Matrix Operation

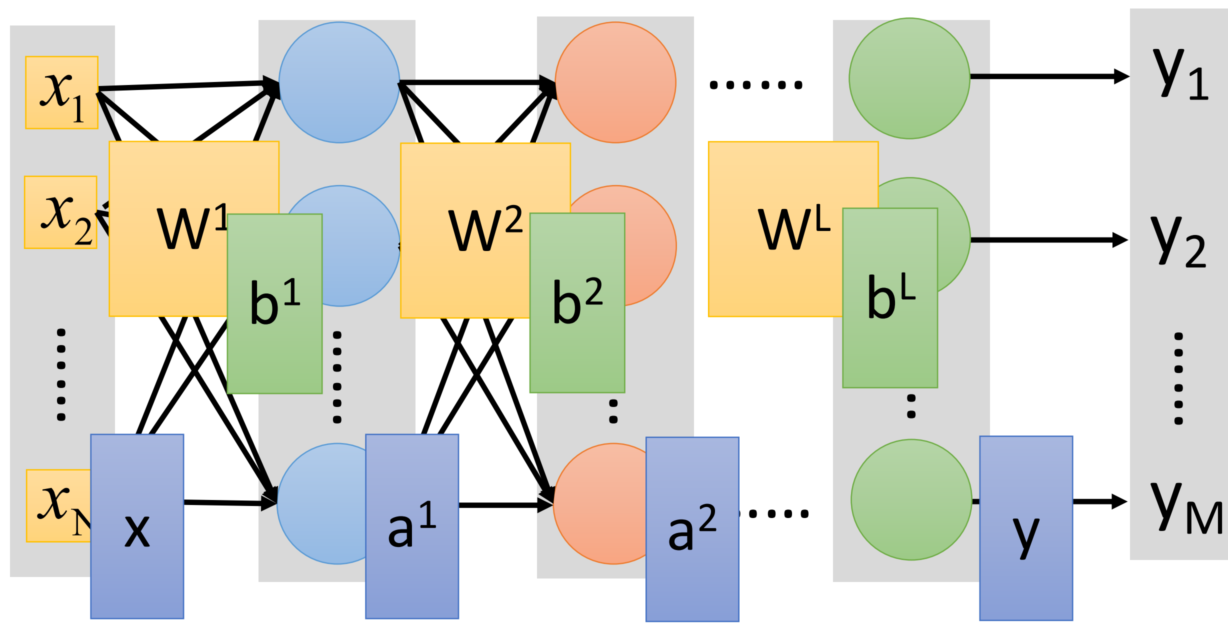



$$\sigma\left( \underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

# Neural Network



# Neural Network




$$y = f(x)$$

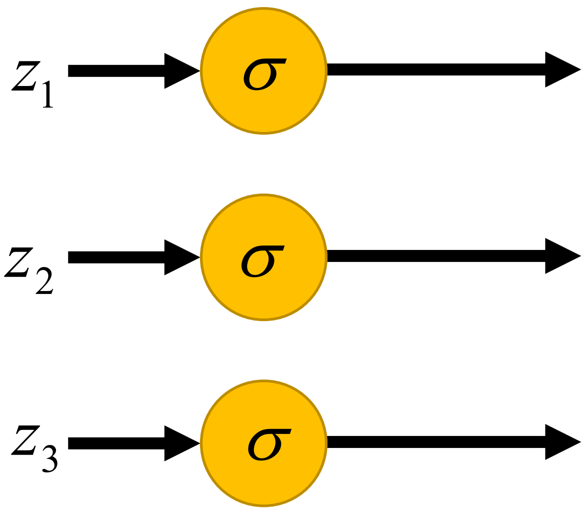
Using parallel computing techniques to speed up matrix operation

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

# Softmax

- Softmax layer as the output layer

***<u>Ordinary Layer</u>***




$$y_1 = \sigma(z_1)$$

$$y_2 = \sigma(z_2)$$

$$y_3 = \sigma(z_3)$$

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

May not be easy to interpret

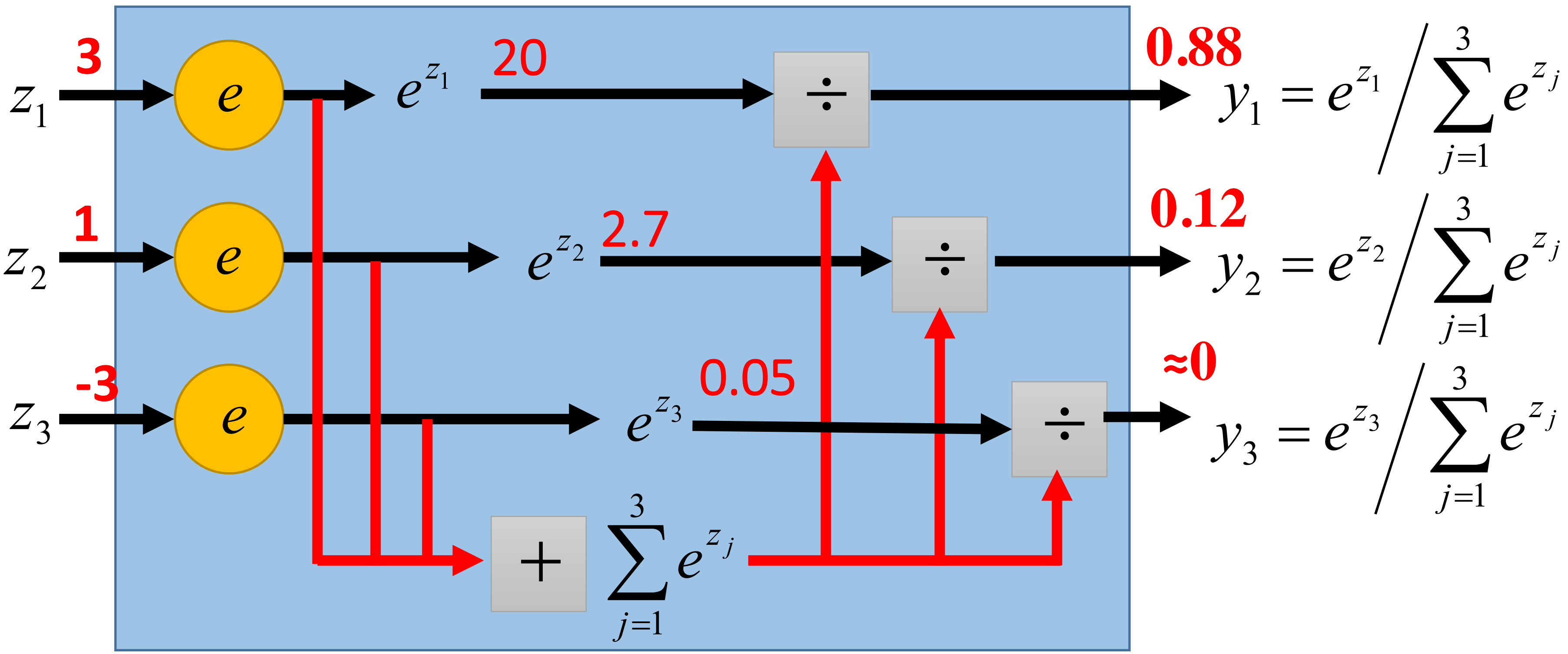
# Softmax

- Softmax layer as the output layer

***Probability***:

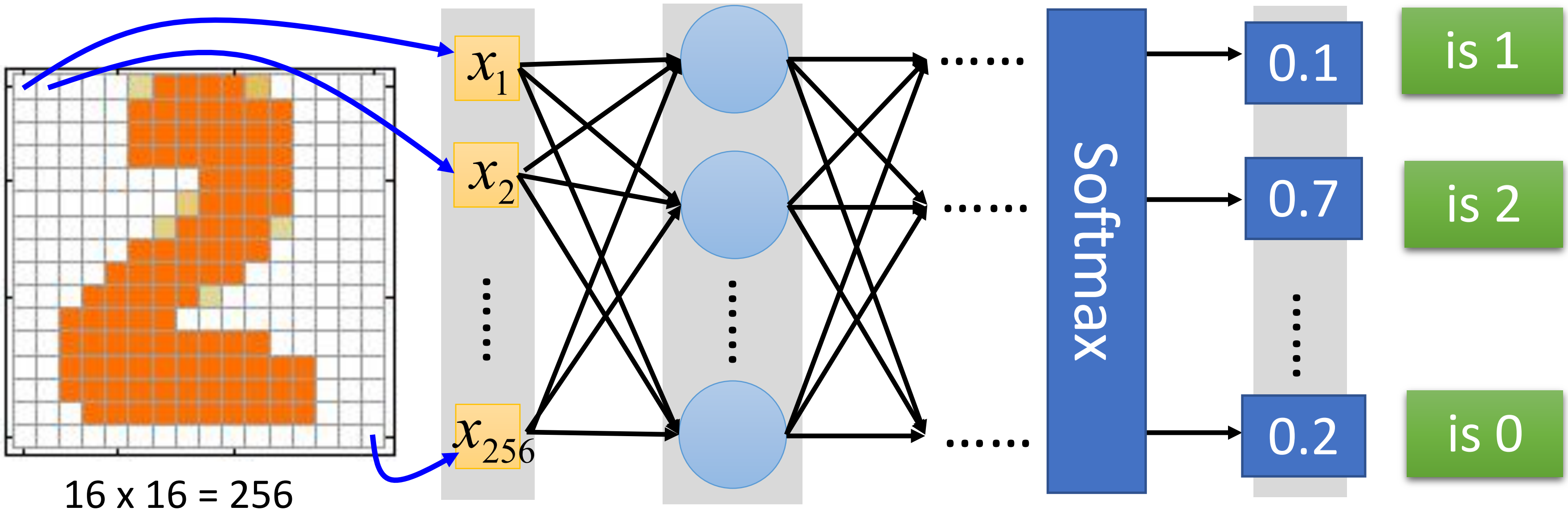
- $1 > y_i > 0$
- $\sum_i y_i = 1$

***Softmax Layer***



# How to set network parameters

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



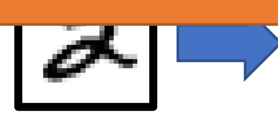

Ink → 1

No ink → 0

Set the network parameters $\theta$ such that ......

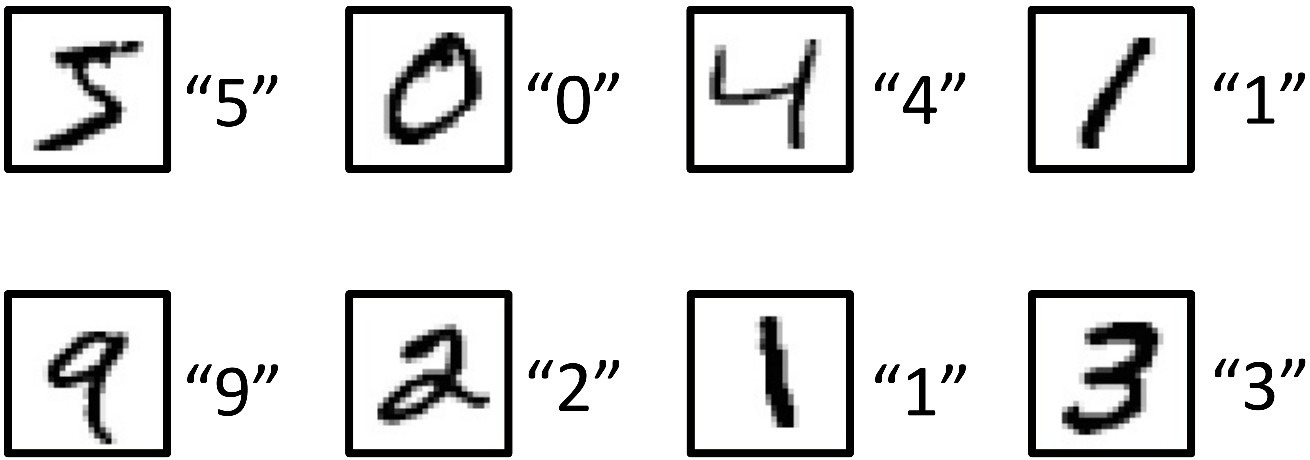
Input m value

How to let the neural network achieve this

Input:  $y_2$ has the maximum value

# Training Data

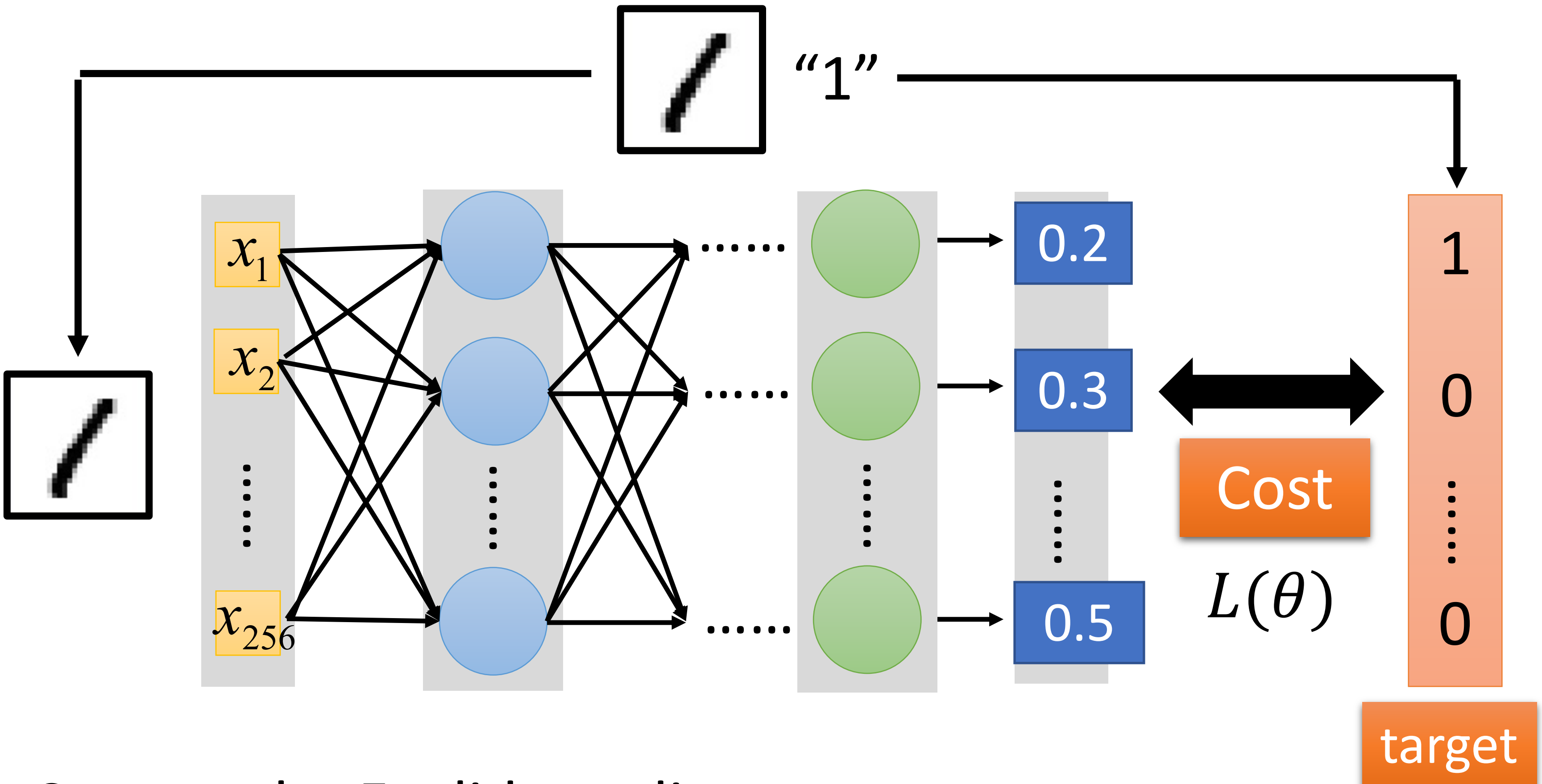
- Preparing training data: images and their labels




Using the training data to find the network parameters.

# Cost

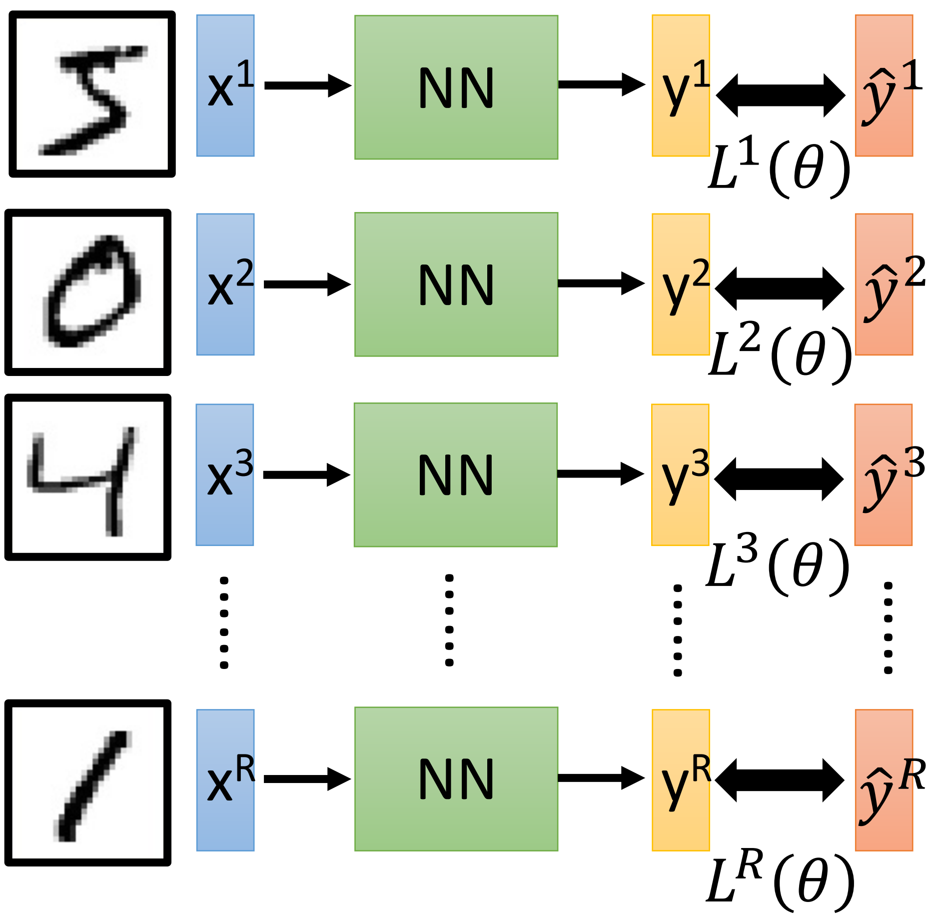
Given a set of network parameters $\theta$, each example has a cost value.




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 $\theta$ is on this task

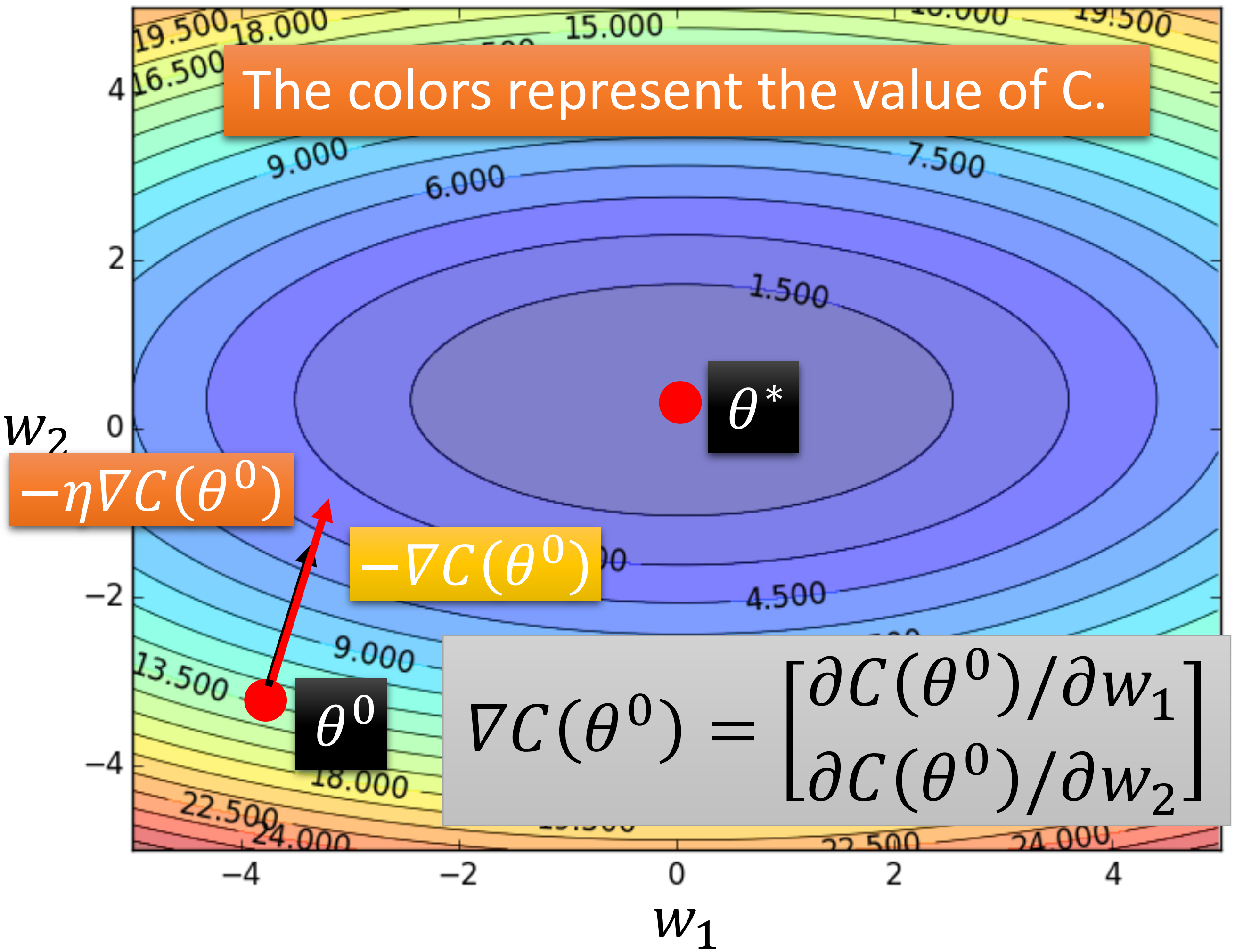
Find the network parameters $\theta^*$ that minimize this value

# Gradient Descent

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

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

Error Surface




Randomly pick a starting point $\theta^0$

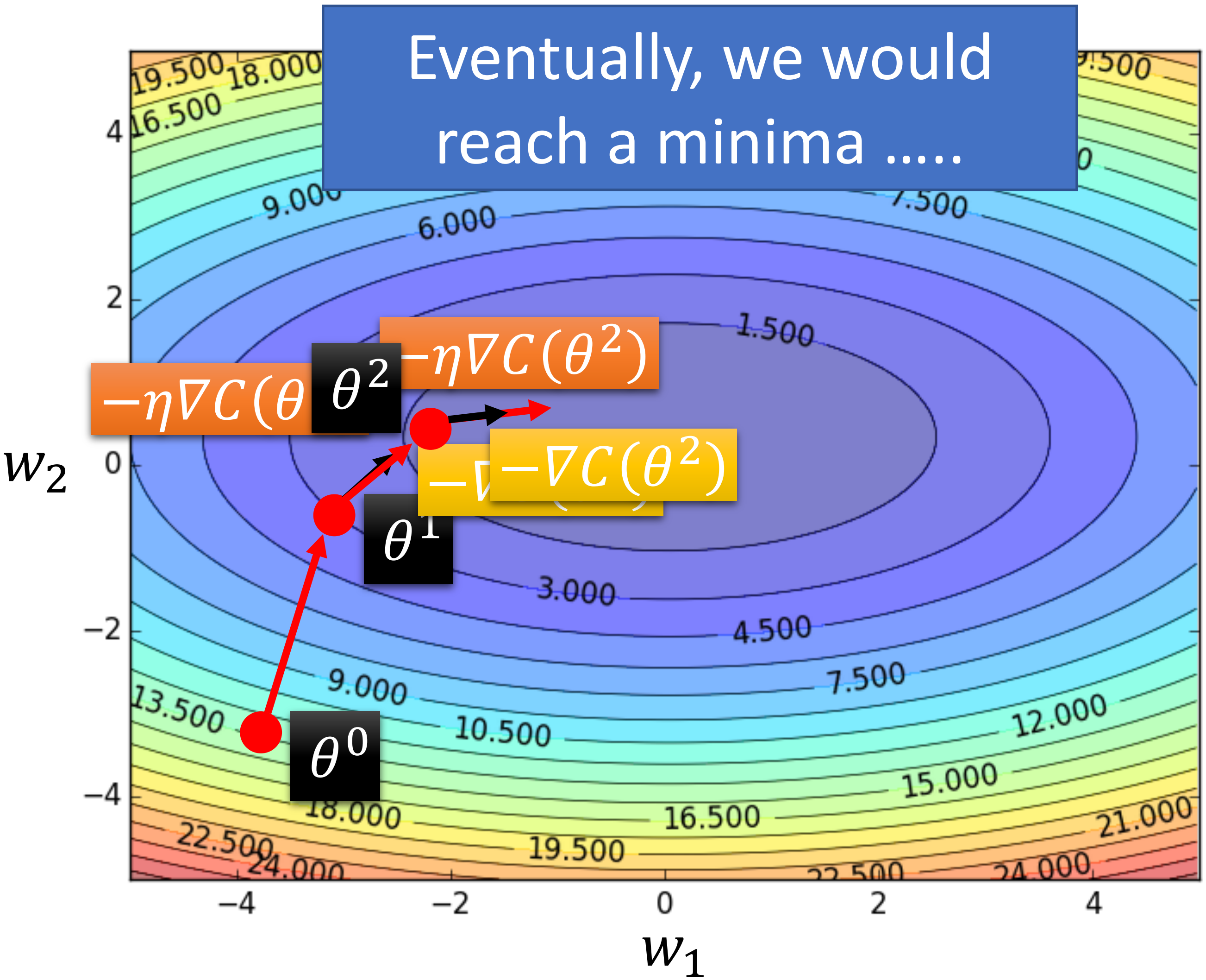
Compute the negative gradient at $\theta^0$

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

Times the learning rate $\eta$

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

# Gradient Descent




Randomly pick a starting point $\theta^0$

Compute the negative gradient at $\theta^0$

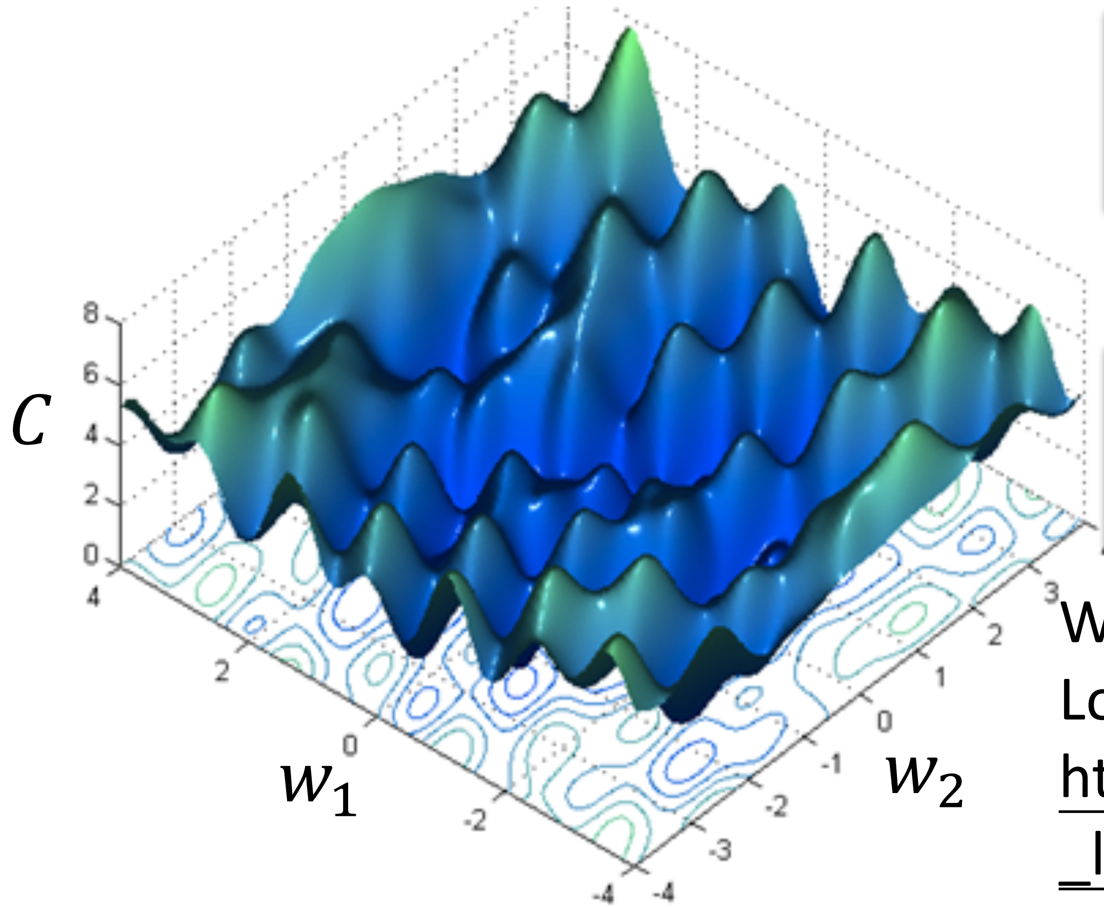
$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate $\eta$

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

# Local Minima

- Gradient descent never guarantee global minima

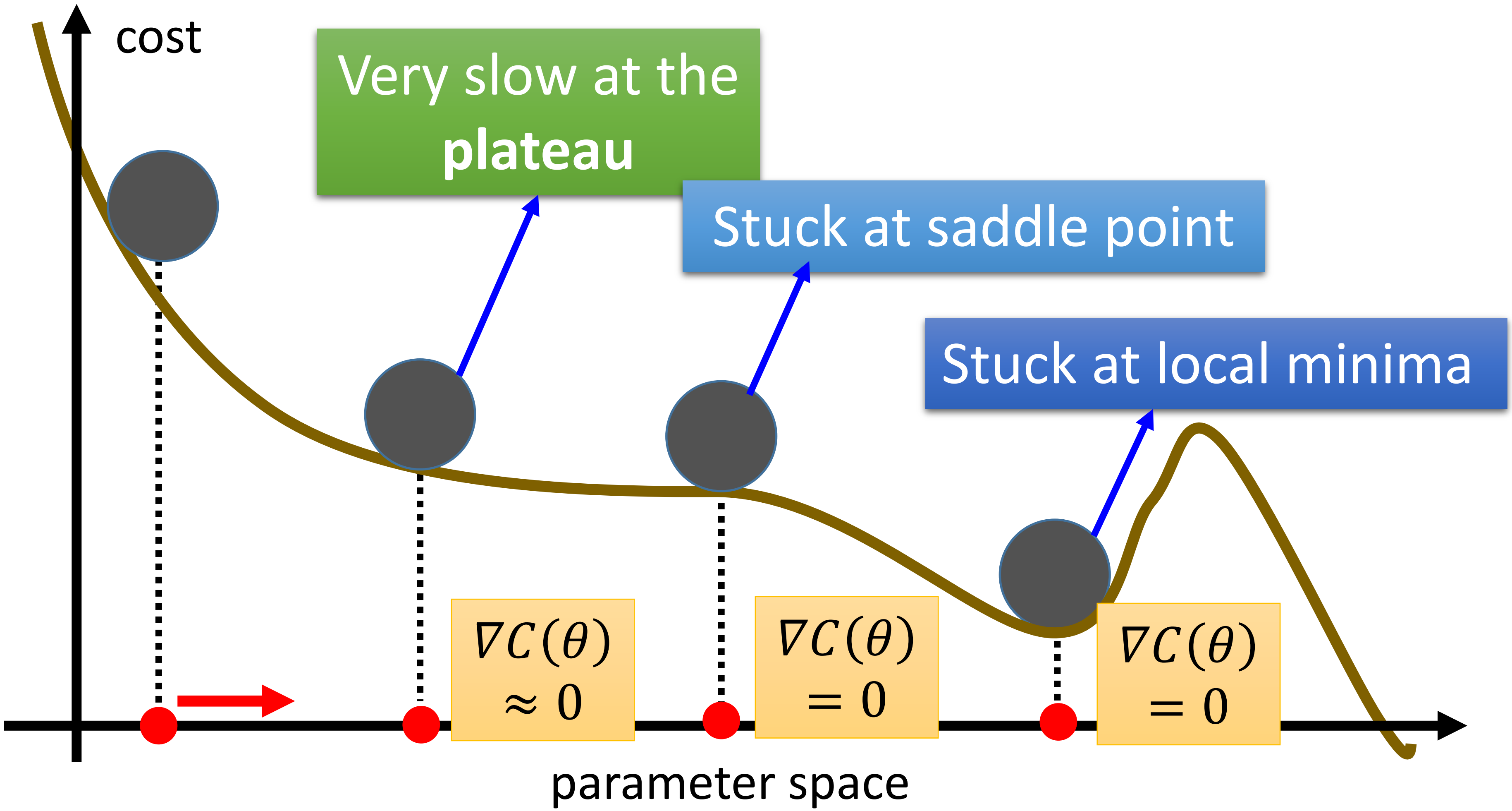



Different initial point $\theta^0$

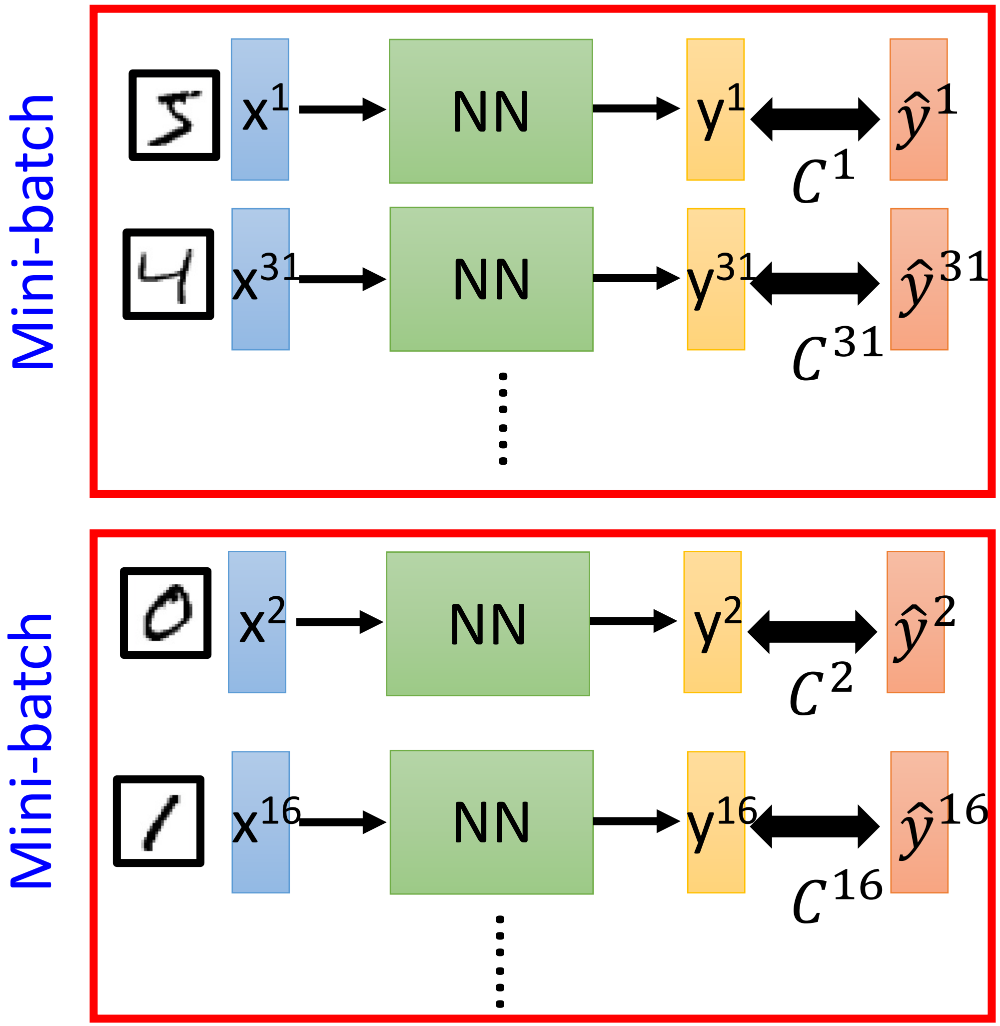
Reach different minima, so different results

Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

# Besides local minima ……



# Mini-batch




- Randomly initialize $\theta^0$
- Pick the $1^{st}$ batch

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

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the $2^{nd}$ batch

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

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

  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

# Neural Networks

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

$$x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D}$$

# Neural Networks

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

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

$$W_2 \in \mathbb{R}^{C \times H} \quad W_1 \in \mathbb{R}^{H \times D} \quad x \in \mathbb{R}^D$$

(In practice we will usually add a learnable bias at each layer as well)

# Neural Networks

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

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

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

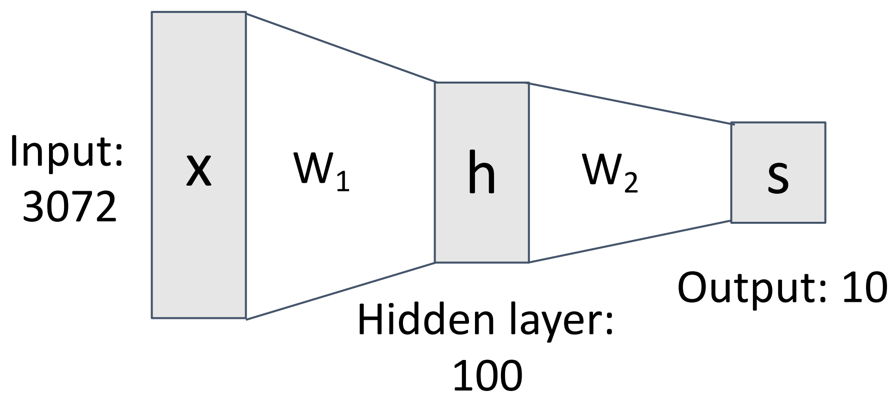
$$W_3 \in \mathbb{R}^{C \times H_2} \quad W_2 \in \mathbb{R}^{H_2 \times H_1} \quad W_1 \in \mathbb{R}^{H_1 \times D} \quad x \in \mathbb{R}^D$$

(In practice we will usually add a learnable bias at each layer as well)

# Neural Networks

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

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



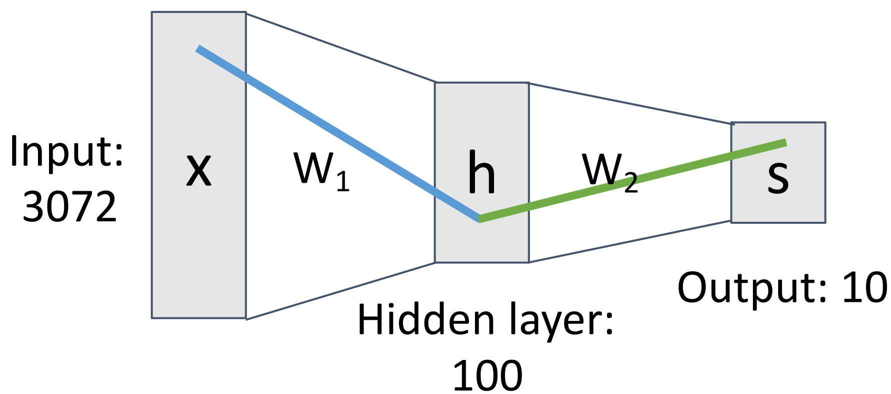

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

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

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

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 $h_j$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

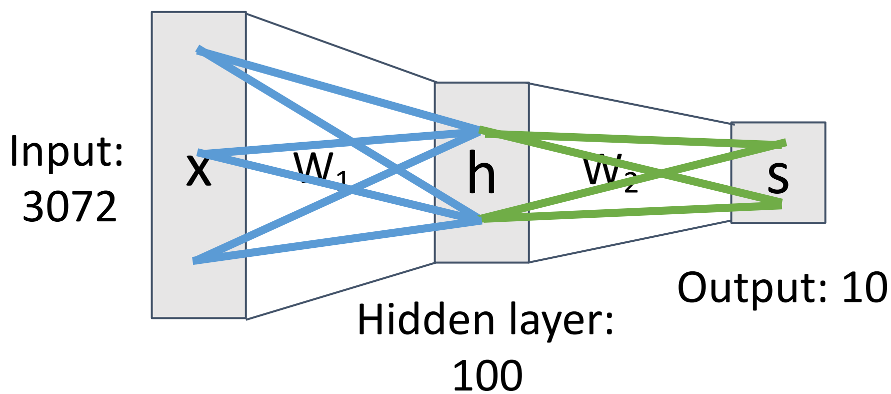
# Neural Networks

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

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

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

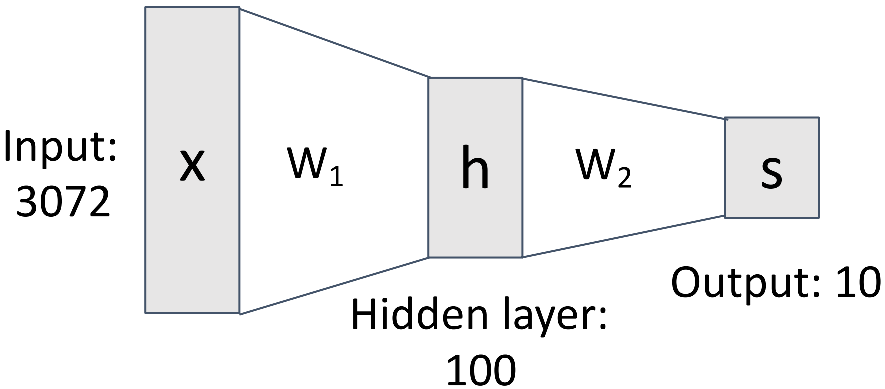
Fully-connected neural network
Also "Multi-Layer Perceptron" (MLP)

# Neural Networks

(**Before**) Linear score function:

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

Linear classifier: One template per class

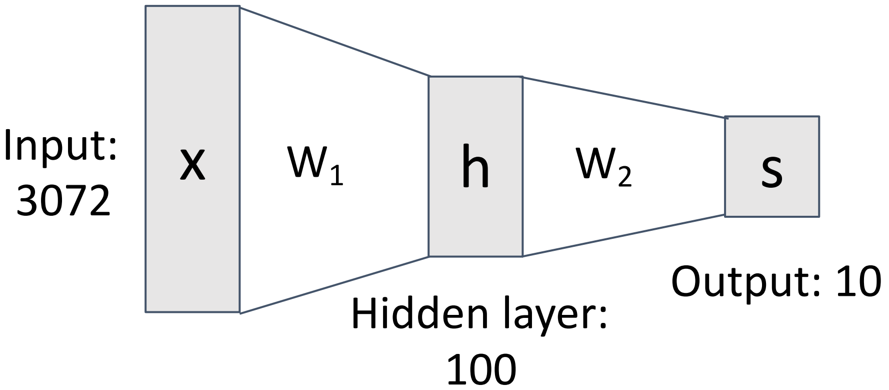

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

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

(**Before**) Linear score function:

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



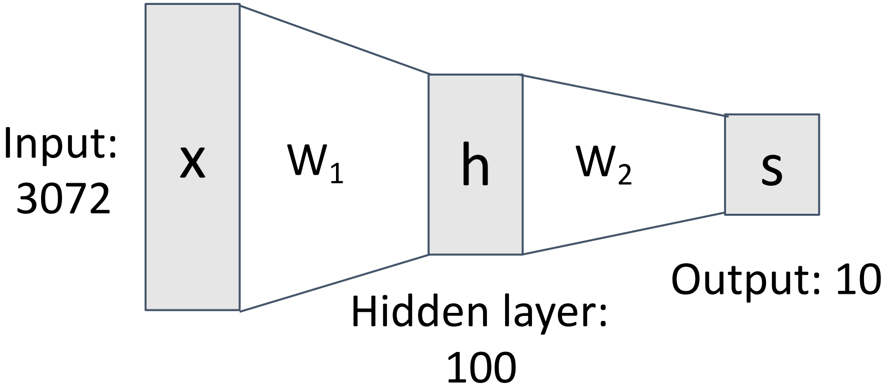

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

Can use different templates to cover multiple modes of a

(**Before**) Linear score function:

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



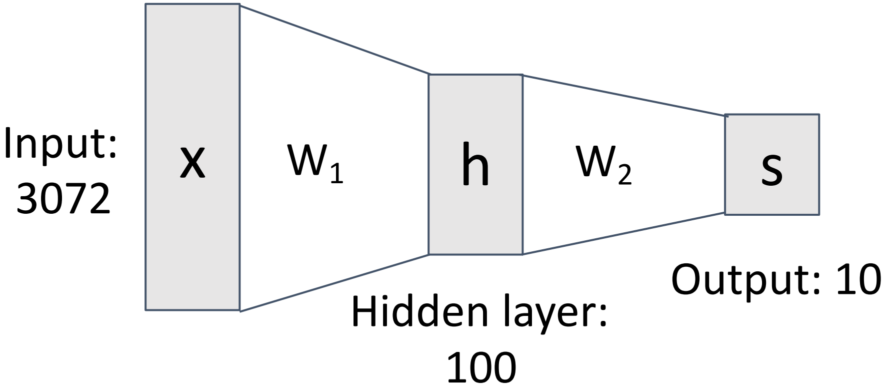

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Neural Networks

"Distributed representation":
Most templates not

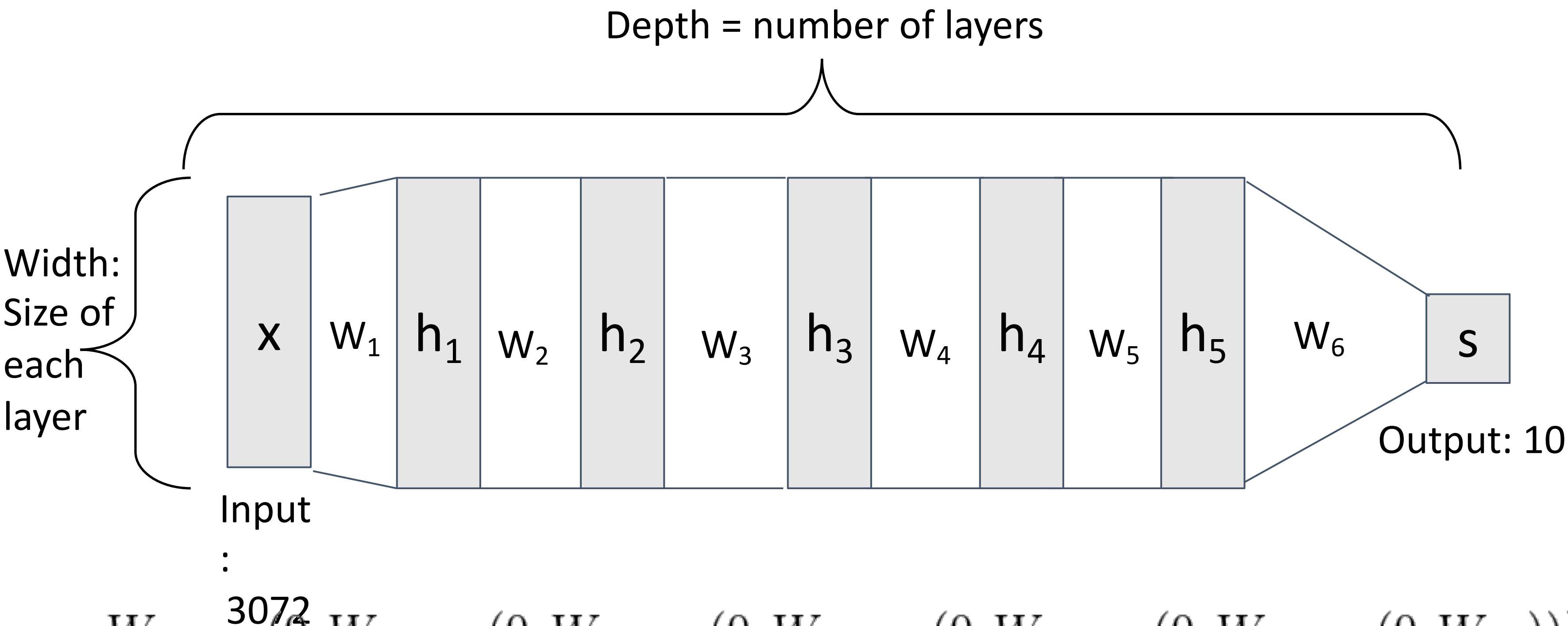
(**Before**) Linear score function:

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




$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Deep Neural Networks




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

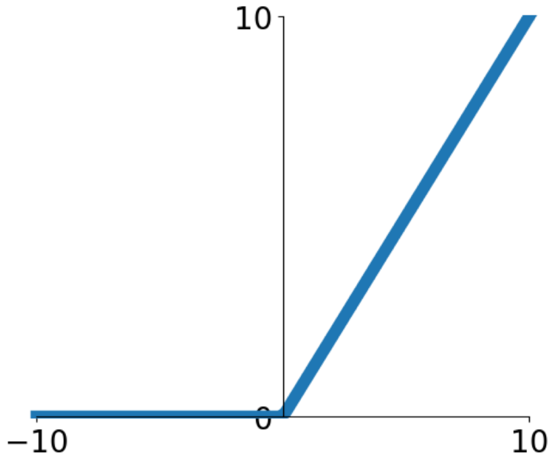
# Activation Functions

2-layer Neural Network

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

The function $ReLU(z) = max(0, z)$ is called "Rectified Linear Unit"

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

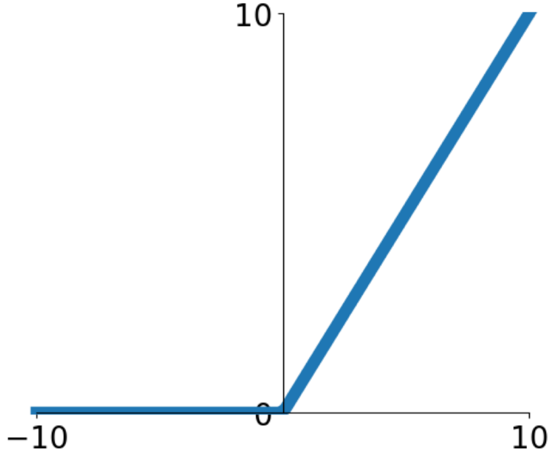


# Activation Functions

2-layer Neural Network

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

The function $ReLU(z) = max(0, z)$ 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?

$$s = W_2 W_1 x$$

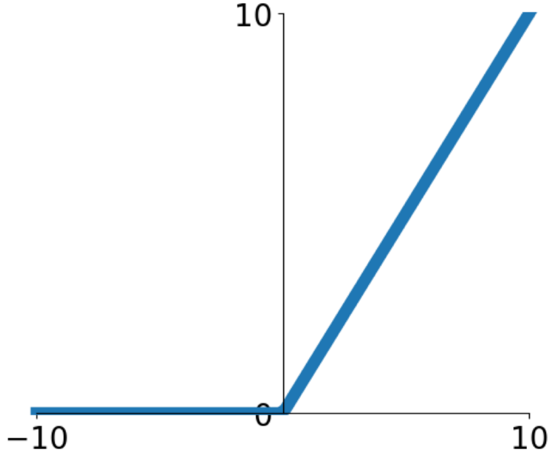
# Activation Functions

2-layer Neural Network

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

The function $ReLU(z) = max(0, z)$ 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?

$$s = W_2 W_1 x$$

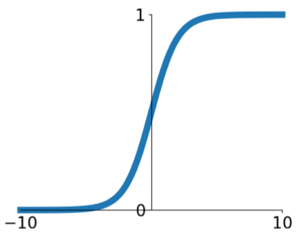
$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times H} \quad s = W_3 x$$

**A**: We end up with a linear classifier!

# Activation Functions

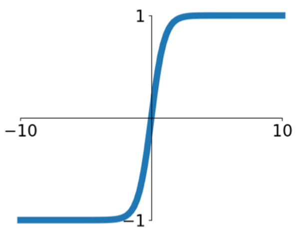
**Sigmoid**

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



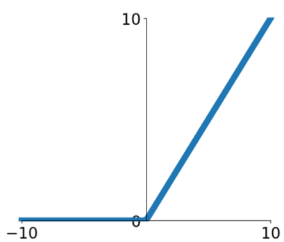

**tanh**

$\tanh(x)$



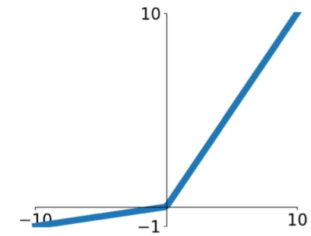

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

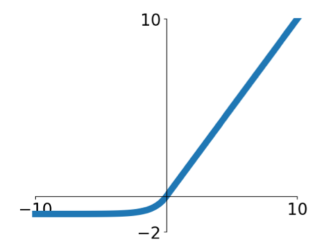



**Maxout**

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

**ELU**

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

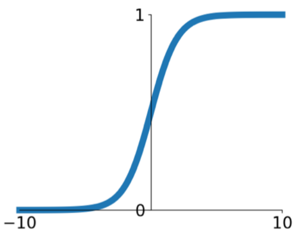


# Activation Functions

ReLU is a good default choice for most problems

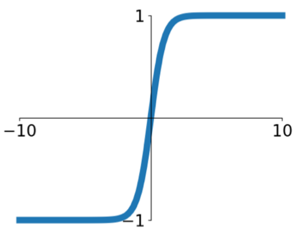
**Sigmoid**

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



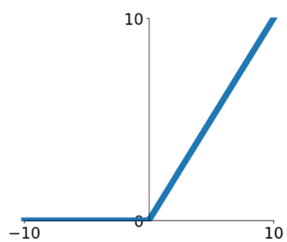

**tanh**

$\tanh(x)$



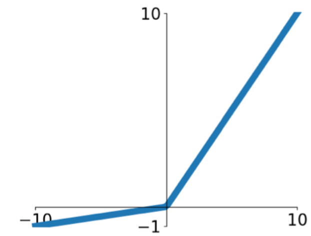

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

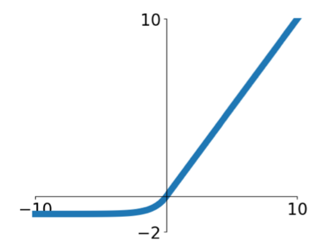



**Maxout**

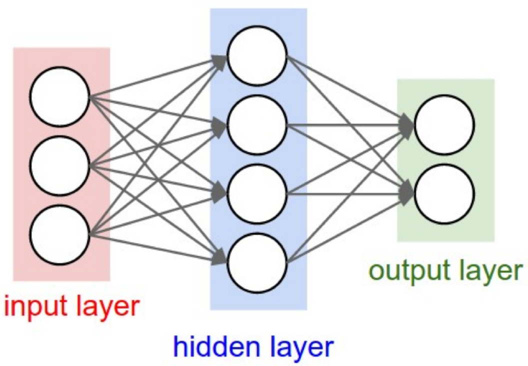
$\max(w_1^T x + b_1, w_2^T x + b_2)$

**ELU**

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



# Neural Net in <20 lines!




Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

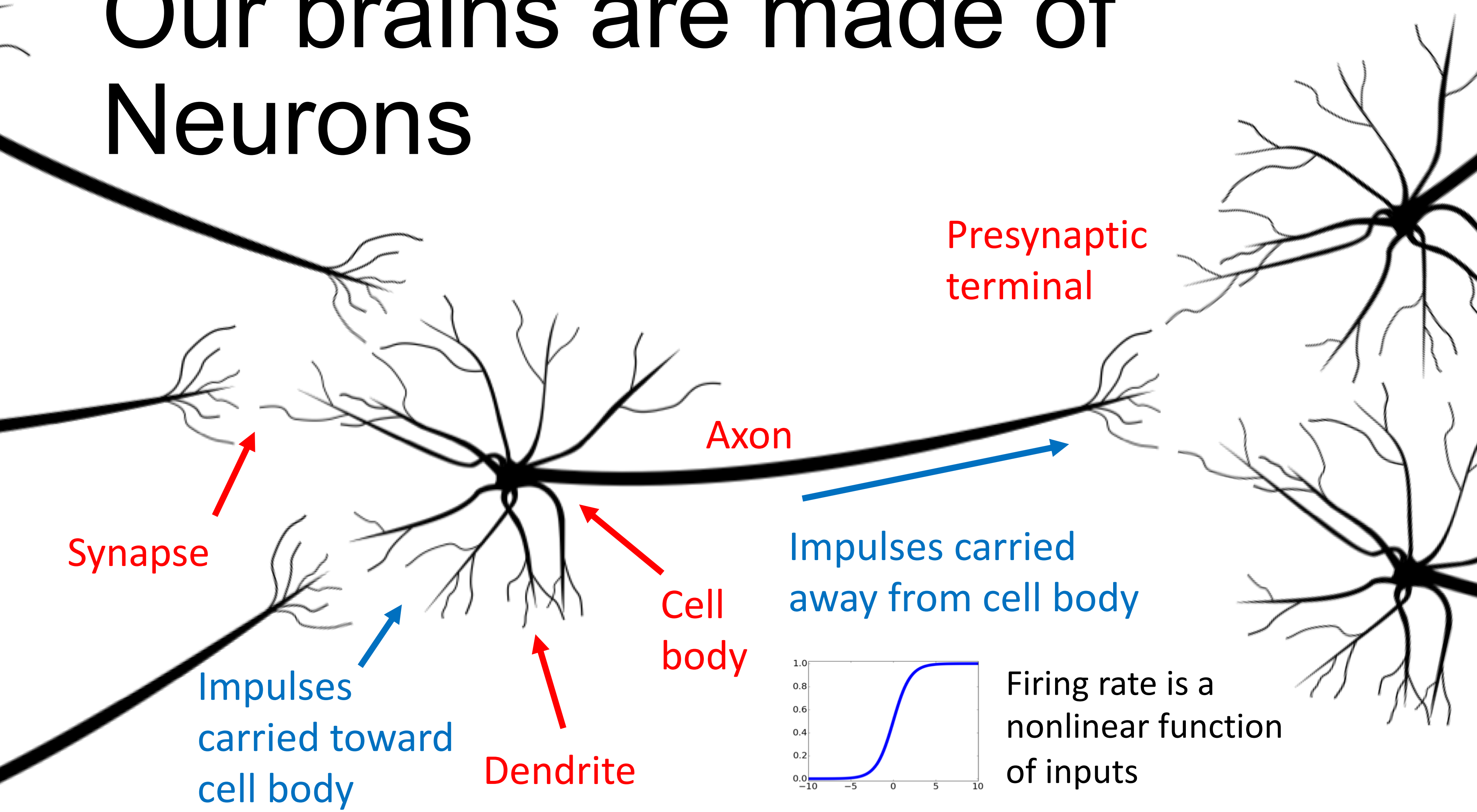
Compute gradients

Stochastic Gradient Descent (SGD) step

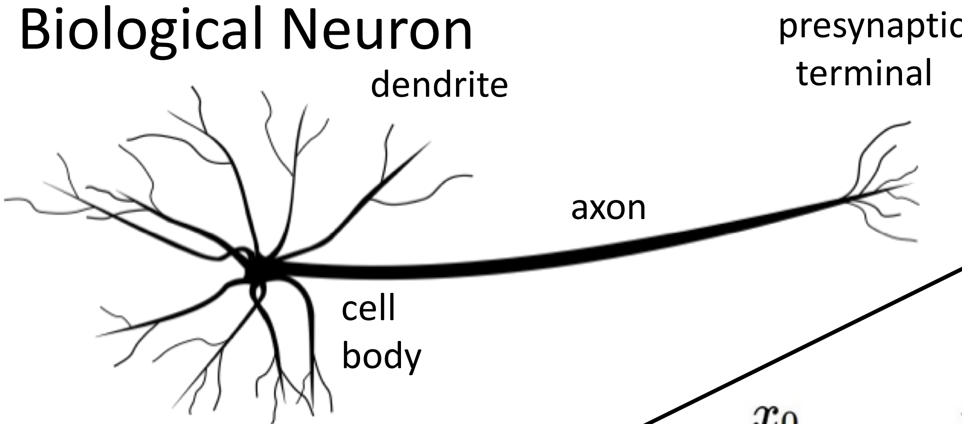
```python
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
for t in range(10000):
  h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  dy_pred = 2.0 * (y_pred - y)
  dw2 = h.T.dot(dy_pred)
  dh = dy_pred.dot(w2.T)
  dw1 = x.T.dot(dh * h * (1 - h))
  w1 -= 1e-4 * dw1
  w2 -= 1e-4 * dw2
```

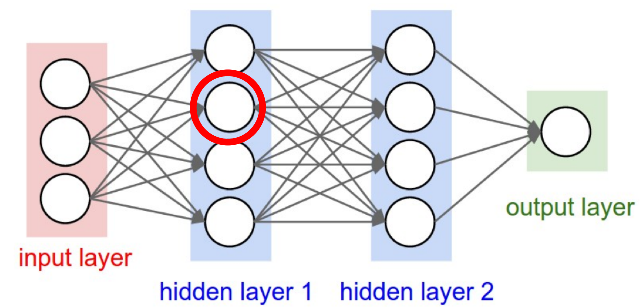
# Our brains are made of Neurons

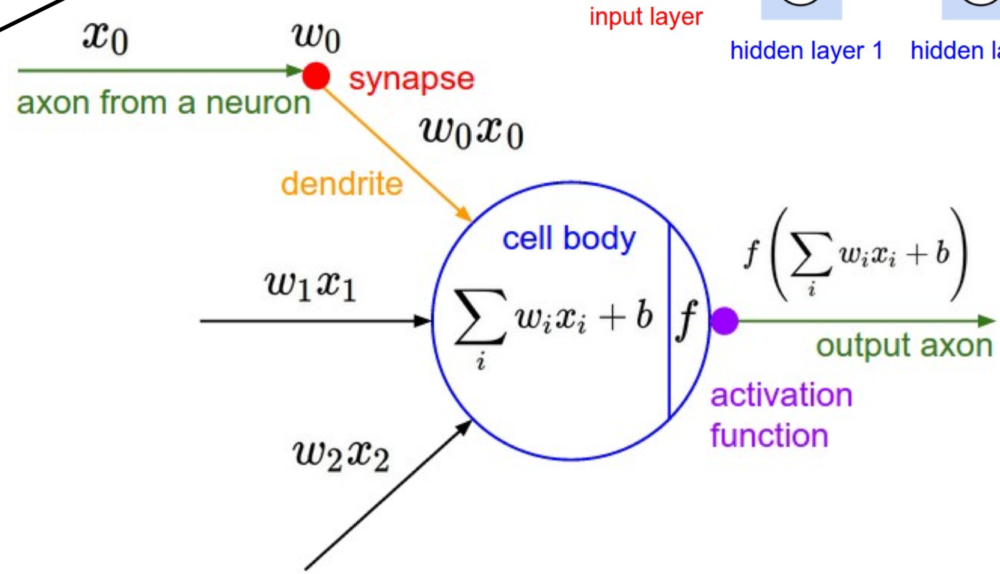


# Biological Neuron

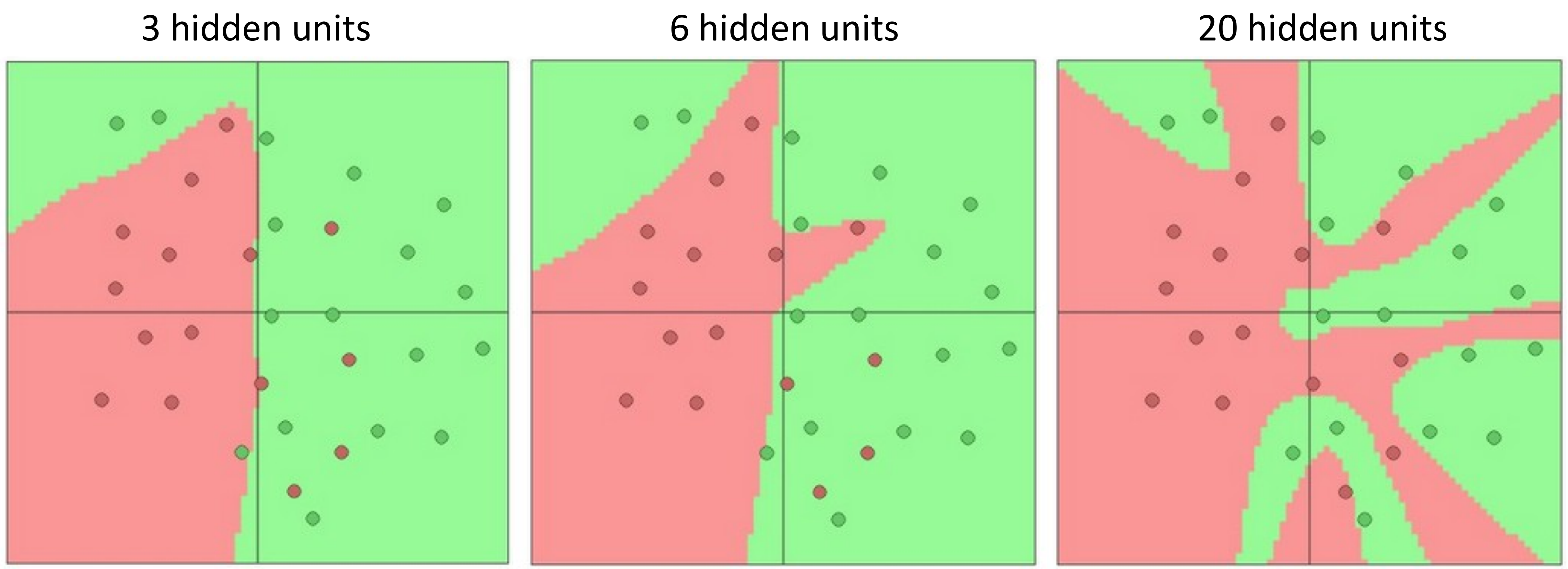



# Artificial Neuron



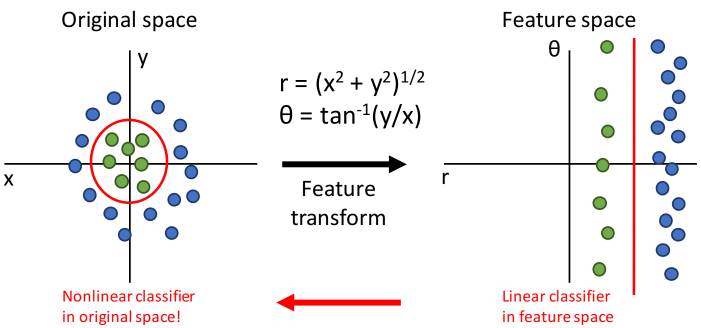
# Setting the number of layers and their sizes



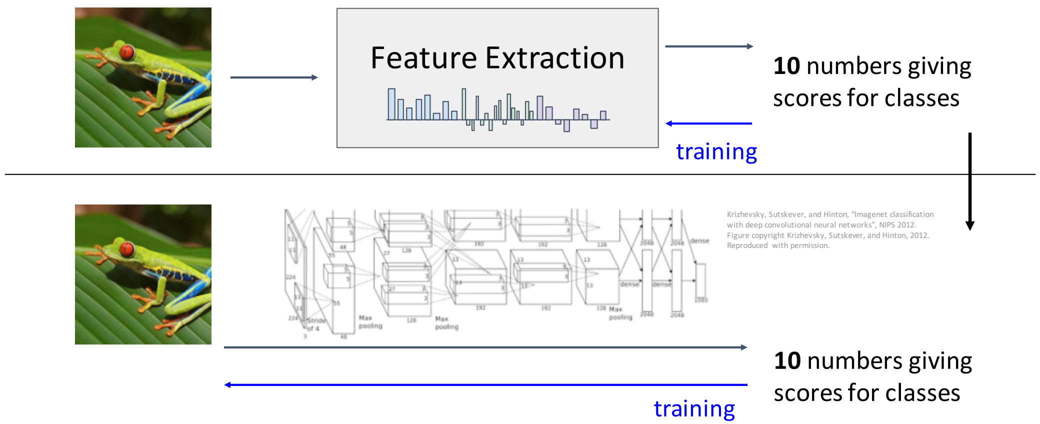

More hidden units = more capacity

# Summary

Feature transform + Linear classifier allows nonlinear decision boundaries



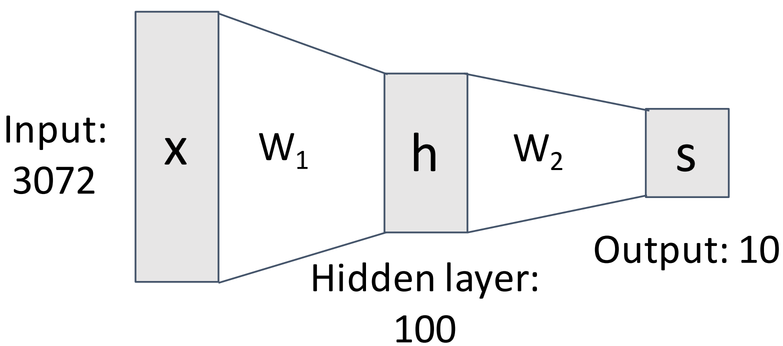

Neural Networks as learnable feature transforms



# Summary

From linear classifiers to
fully-connected networks

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




Linear classifier: One template per class


Neural networks: Many reusable templates

# 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

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

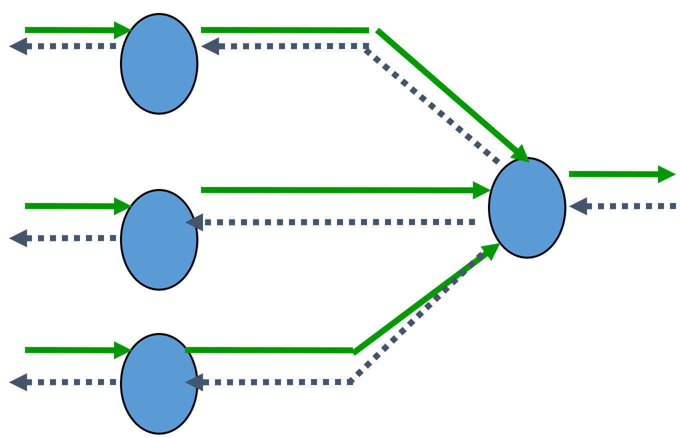
# Training: Backprop algorithm

- 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

*Network activation*
*Forward Step*



*Error propagation*
*Backward Step*

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

# Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

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

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn $W_1$ and $W_2$

# (Bad) Idea: Derive on paper $\nabla_W L$

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

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

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

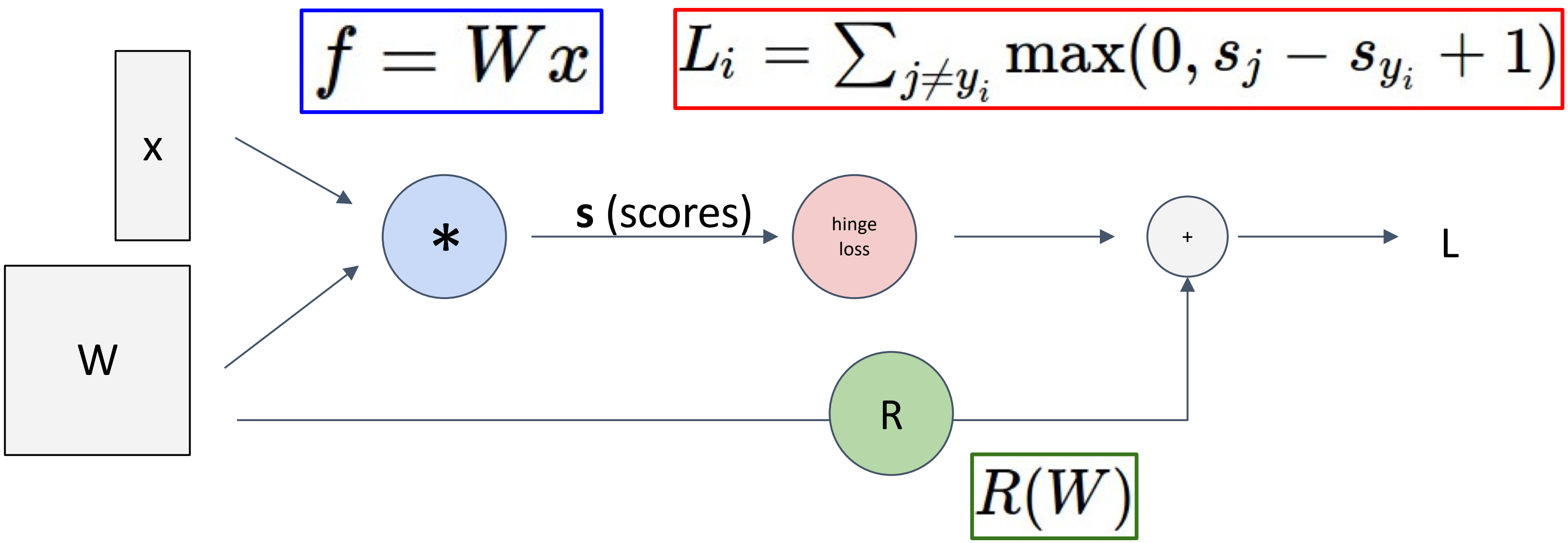
$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

**Problem**: Very tedious: Lots of matrix calculus, need lots of paper

**Problem**: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch. Not modular!

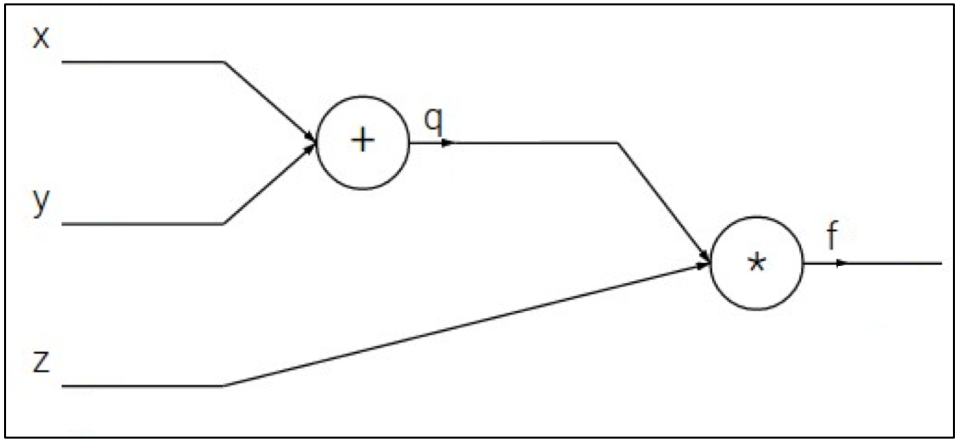
**Problem**: Not feasible for very complex models!

# Better Idea: **Computational Graphs**



# Backpropagation: Simple Example

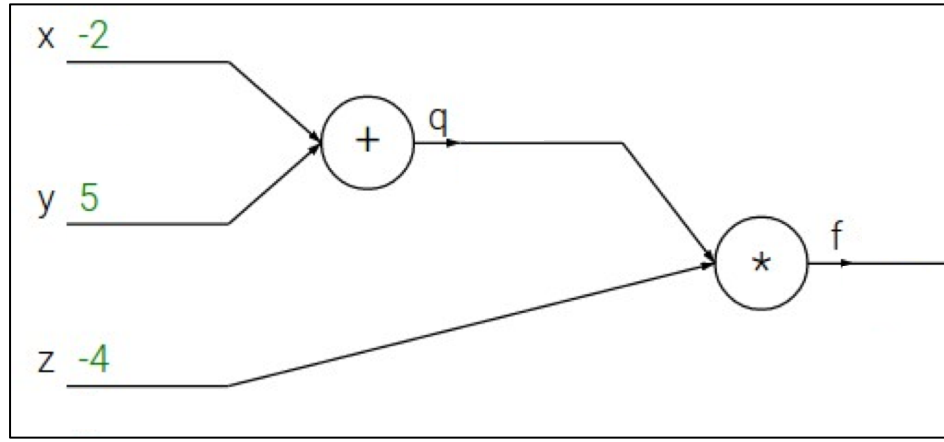
$$f(x, y, z) = (x + y)z$$



# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

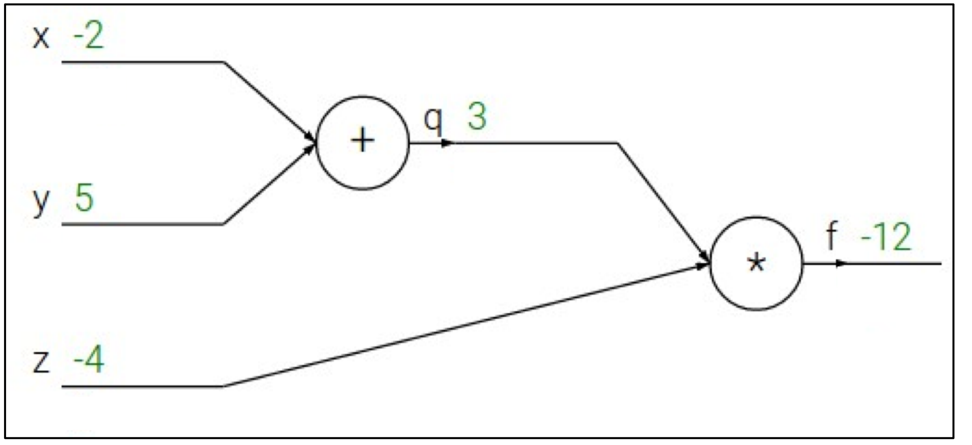
e.g. x = -2, y = 5, z = -4



# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

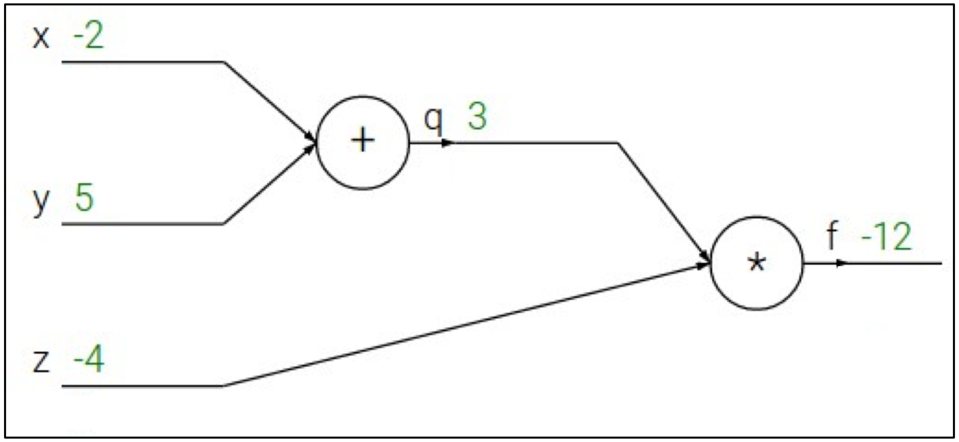
e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

# Backpropagation: Simple Example




$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

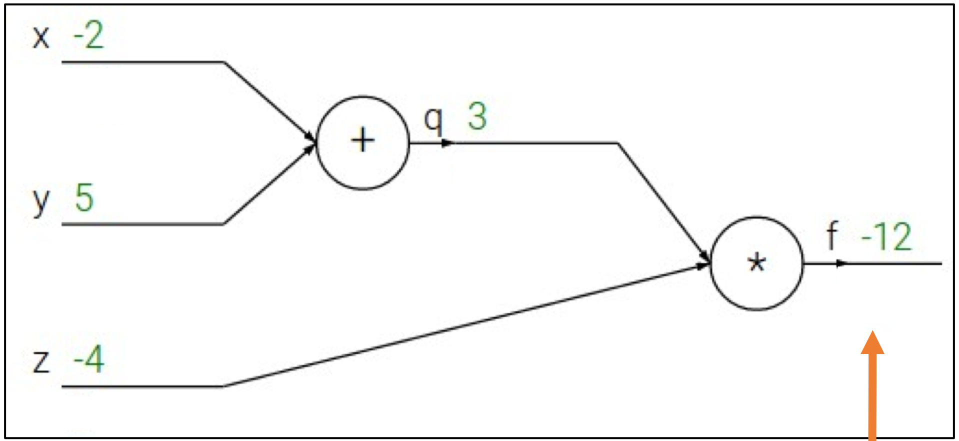
**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




$$\boxed{\frac{\partial f}{\partial f}}$$

**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

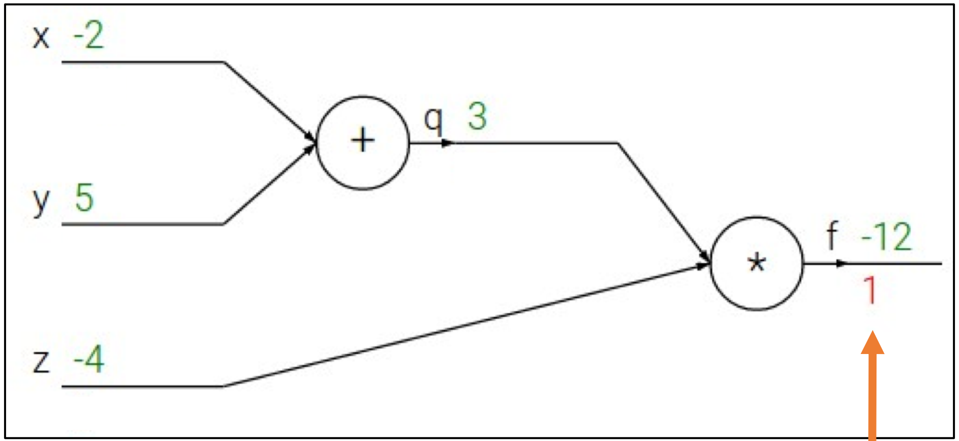
**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




$$\boxed{\frac{\partial f}{\partial f}}$$

**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

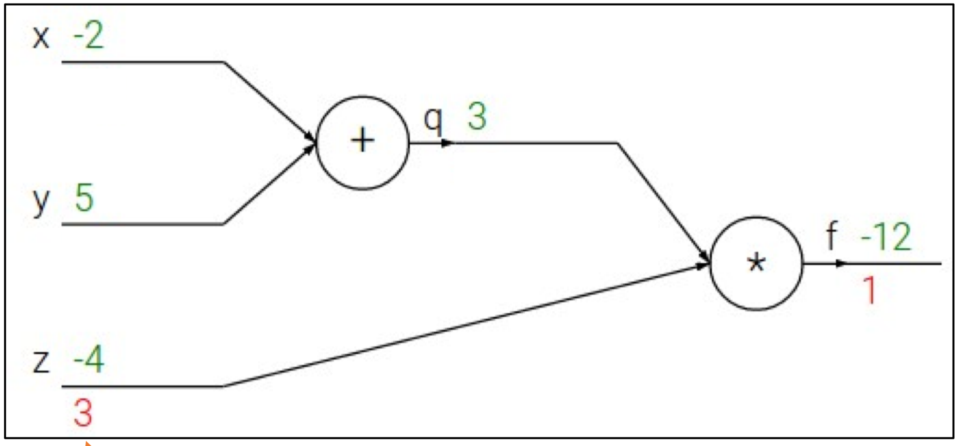
**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$q = x + y \qquad \boxed{f = qz}$$

**2. Backward pass**: Compute derivatives

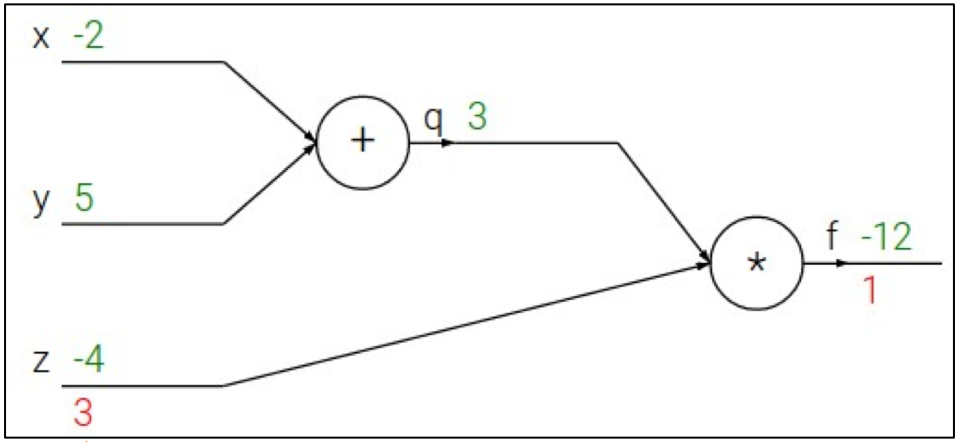
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial z}}$$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$q = x + y \qquad \boxed{f = qz}$$

**2. Backward pass**: Compute derivatives

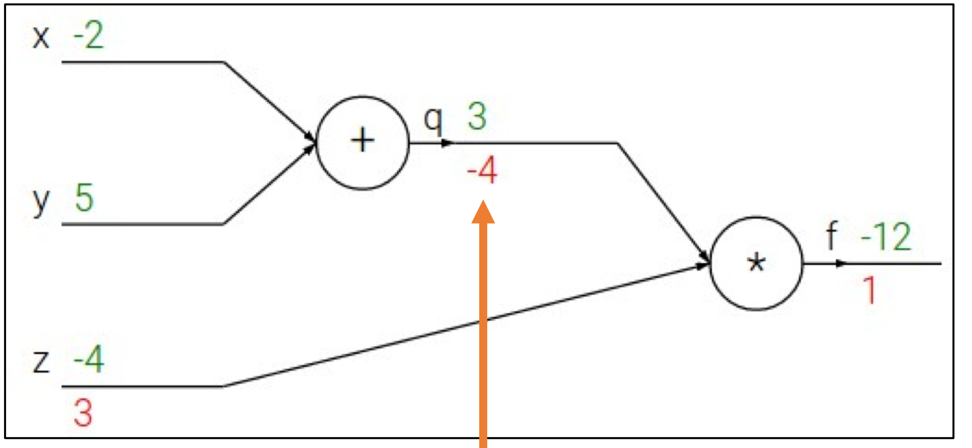
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\boxed{\frac{\partial f}{\partial z} = q}$$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




$$\boxed{\frac{\partial f}{\partial q}}$$

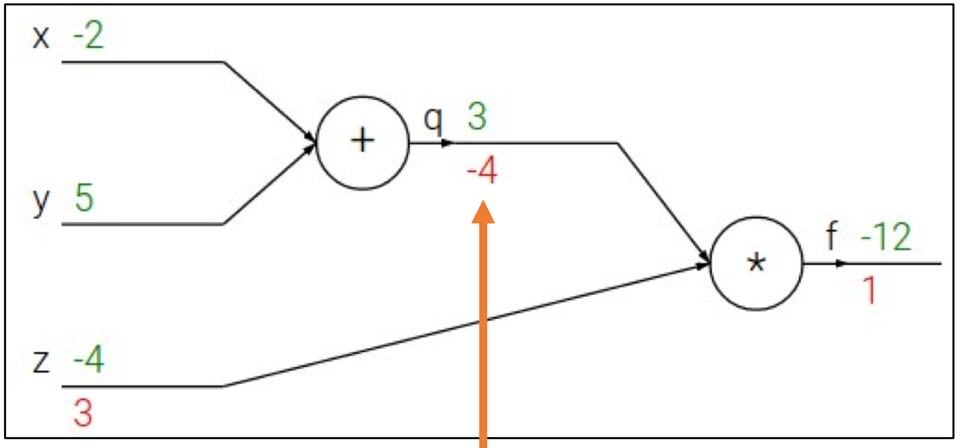
**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example




$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

**1. Forward pass**: Compute outputs

$$q = x + y \qquad \boxed{f = qz}$$

$$\boxed{\frac{\partial f}{\partial q} = z}$$

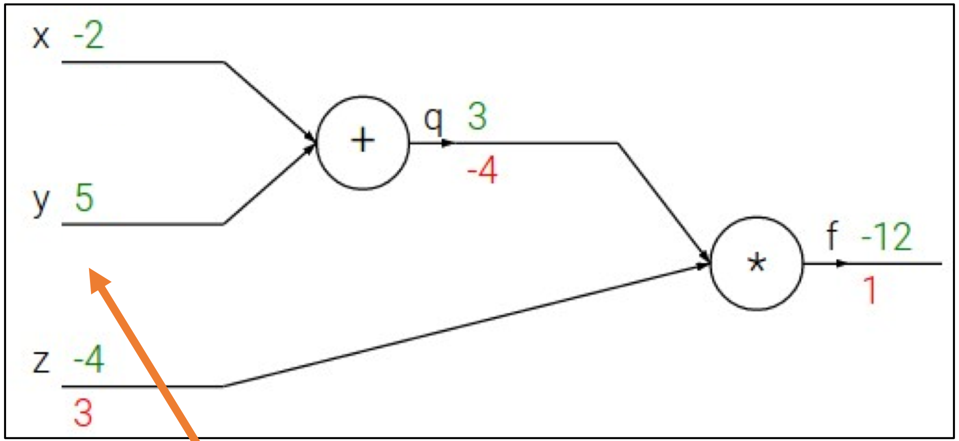
**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

$$\boxed{\frac{\partial f}{\partial y}}$$

**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

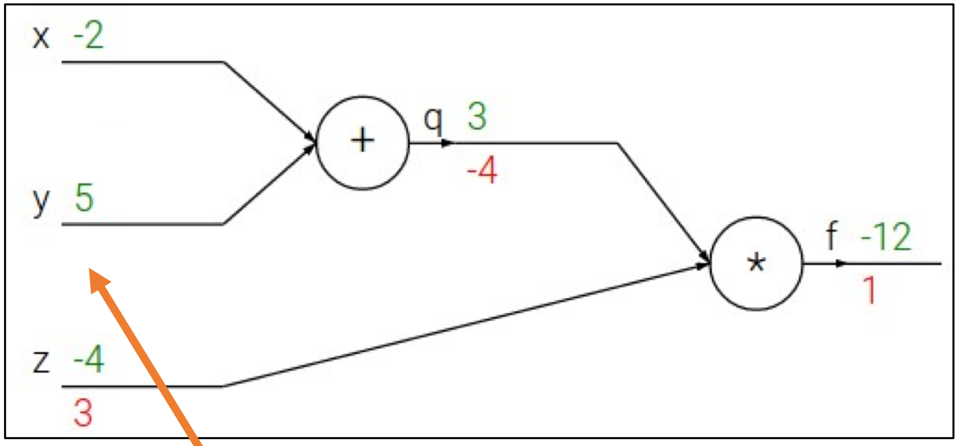
e.g. x = -2, y = 5, z = -4

**1. Forward pass**: Compute outputs

$$q = x + y \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



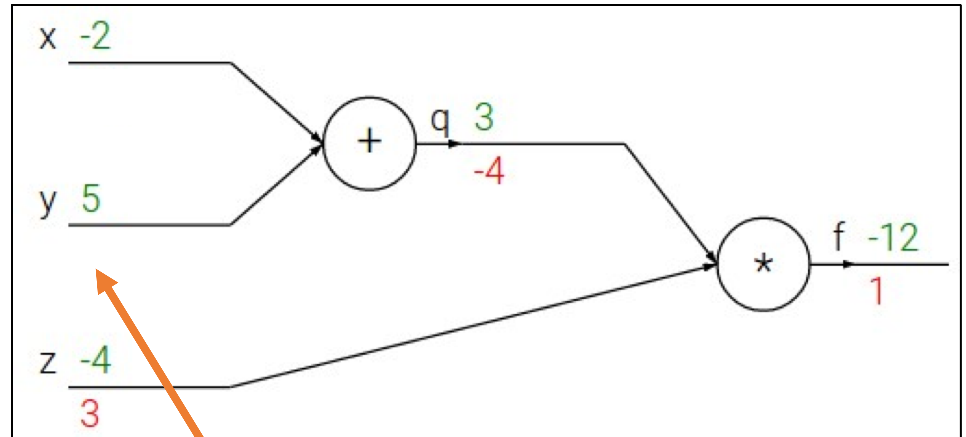

**Chain Rule**

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}}$$

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

**Chain Rule**

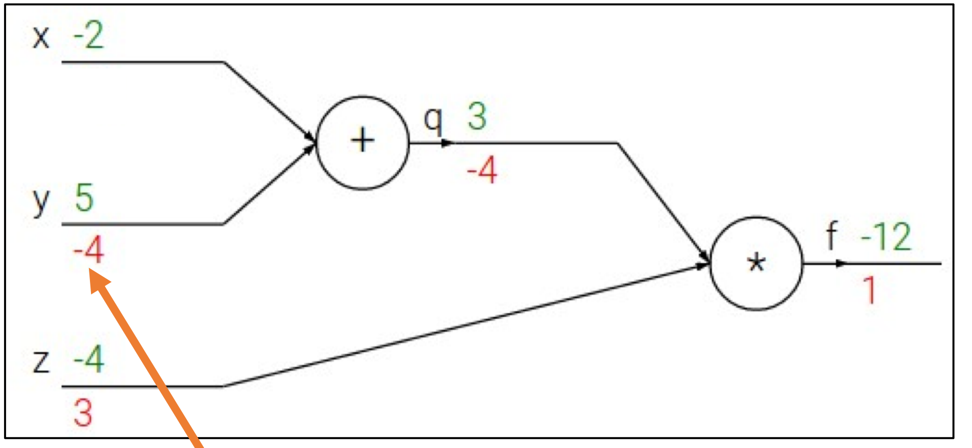
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y} \frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

**Downstream Gradient** **Local Gradient** **Upstream Gradient**

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

**Chain Rule**

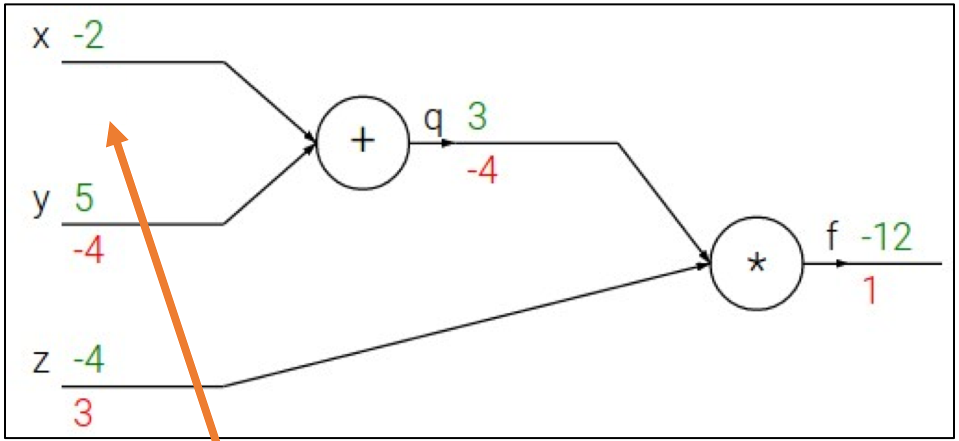
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial y} = 1}$$

**Downstream Gradient** **Local Gradient** **Upstream Gradient**

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4




**1. Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$

**Chain Rule**

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

**Downstream Gradient** **Local Gradient** **Upstream Gradient**

# Backpropagation: Simple Example

$$f(x, y, z) = (x + y)z$$

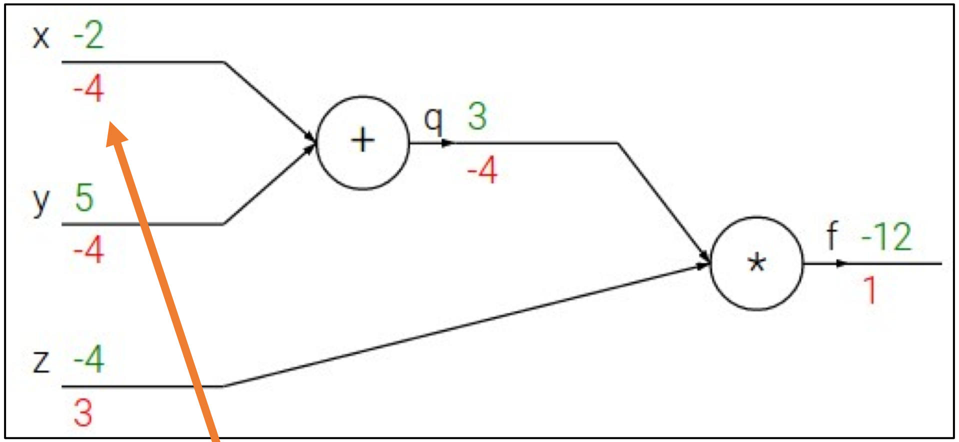
e.g. x = -2, y = 5, z = -4

**1. Forward pass**: Compute outputs

$$\boxed{q = x + y} \qquad f = qz$$

**2. Backward pass**: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




**Chain Rule**

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}} \qquad \boxed{\frac{\partial q}{\partial x} = 1}$$

**Downstream Gradient** **Local Gradient** **Upstream Gradient**

# Part II:
# Why Deep?

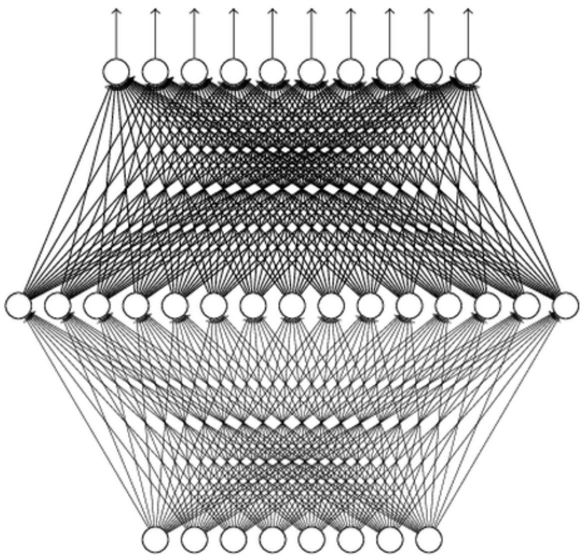
# Universality Theorem

Any continuous function f

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

Can be realized by a network with one hidden layer

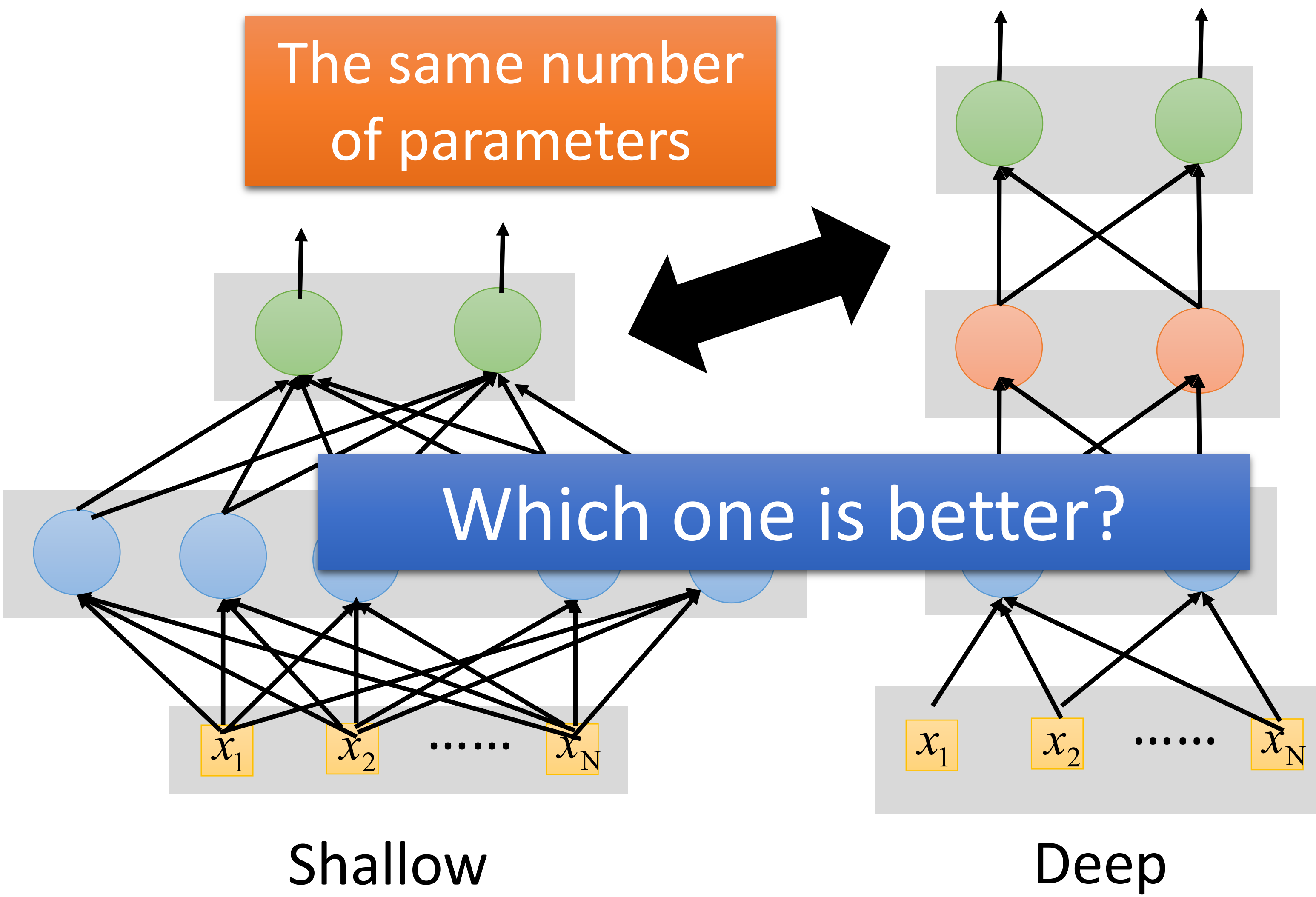
(given **enough** hidden neurons)



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

Why "Deep" neural network not "Fat" neural network?

# Fat + Short v.s. Thin + Tall



Both shallow (a) and deep (b) networks are universal, that is they can approximate arbitrarily well any continuous function of d variables on a compact domain.

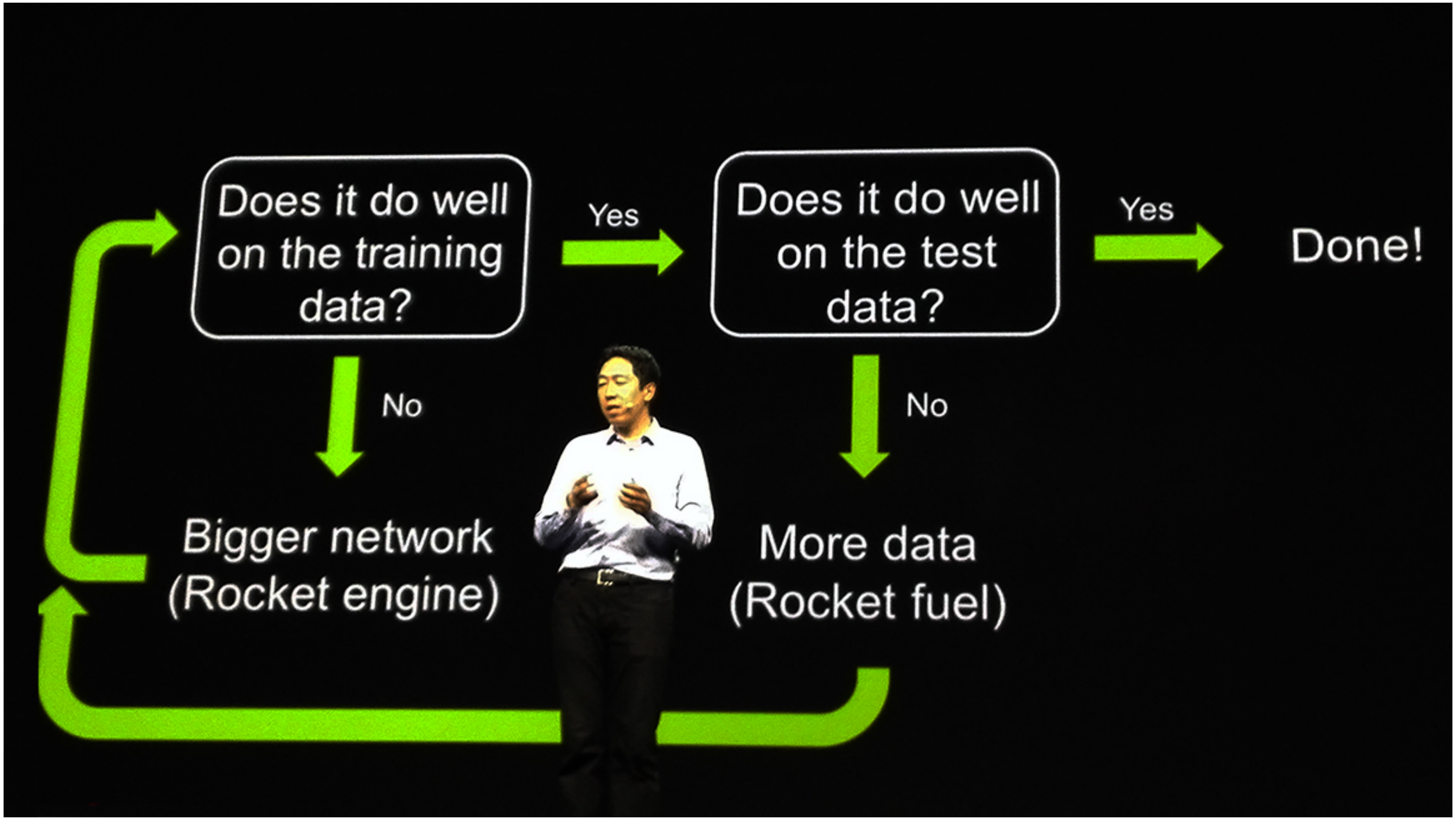
We show that the approximation of functions with a compositional structure – such as $f(x1, \cdots, xd) = h1(h2 \cdots (hj\ (hi1(x1, x2), hi2(x3, x4)), \cdots))$ – can be achieved with the same degree of accuracy by deep and shallow networks but that the number of parameters, the VC-dimension and the fat-shattering dimension are much smaller for the deep networks than for the shallow network with equivalent approximation accuracy.

It is intuitive that a hierarchical network matching the structure of a compositional function should be “better” at approximating it than a generic shallow network but universality of shallow networks makes the statement less than obvious. Our result makes clear that the intuition is indeed correct and provides quantitative bounds.

Why are compositional functions important? We argue that the basic properties of scalability and shift invariance in many natural signals such as images and text require compositional algorithms that can be well approximated by Deep Convolutional Networks. Of course, there are many situations that do not require shift invariant, scalable algorithms. For the many functions that are not compositional we do not expect any advantage of deep convolutional networks.

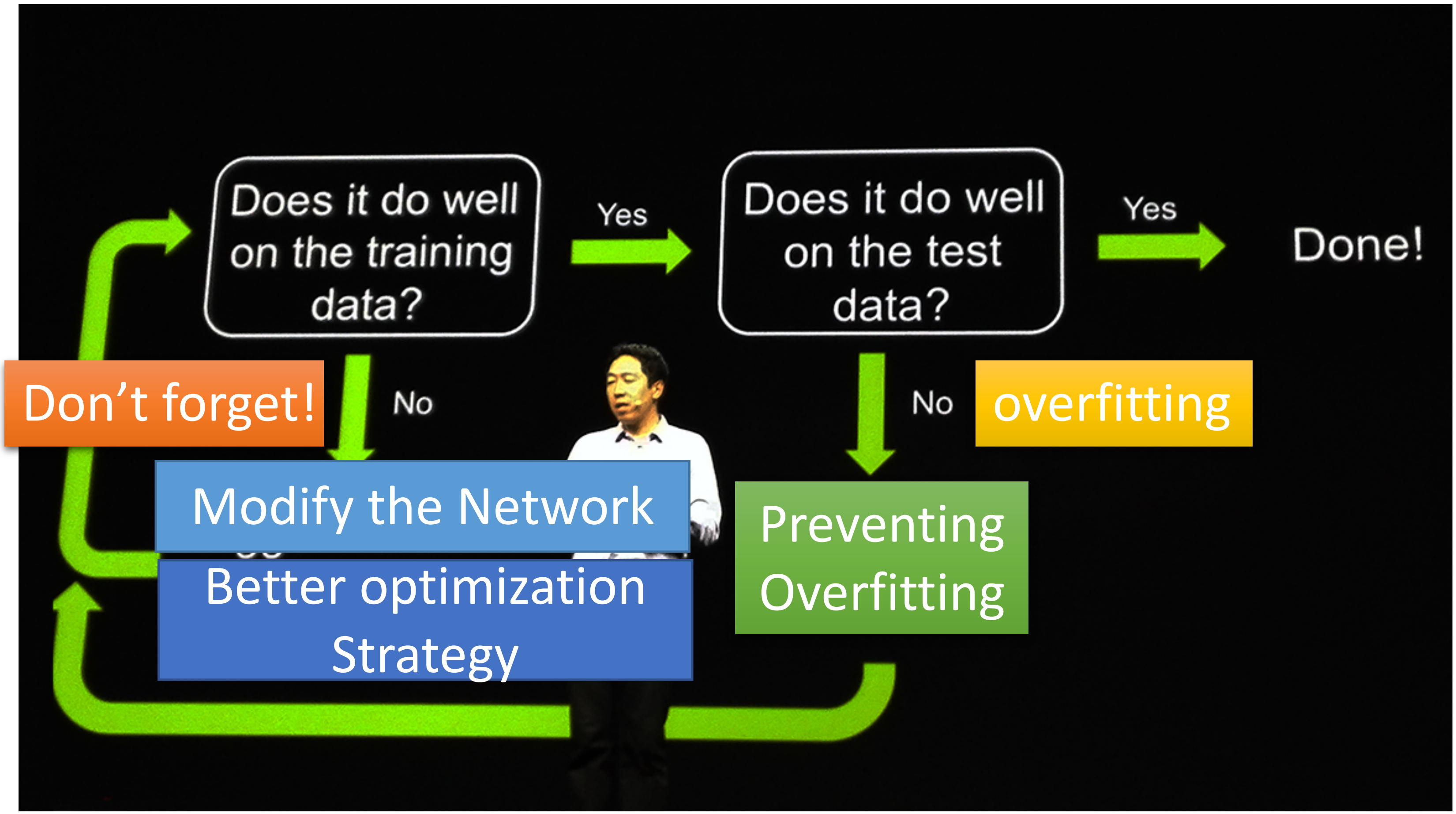
Learning Functions: When Is Deep Better Than Shallow by Hrushikesh Mhaskar Department of Mathematics, California Institute of Technology, Pasadena, CA 91125; Institute of Mathematical Sciences, Claremont Graduate University, Claremont, CA 91711, Qianli Liao and Tomaso Poggio Center for Brains, Minds, and Machines, McGovern Institute for Brain Research Massachusetts Institute of Technology, Cambridge, MA, 02139

# 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

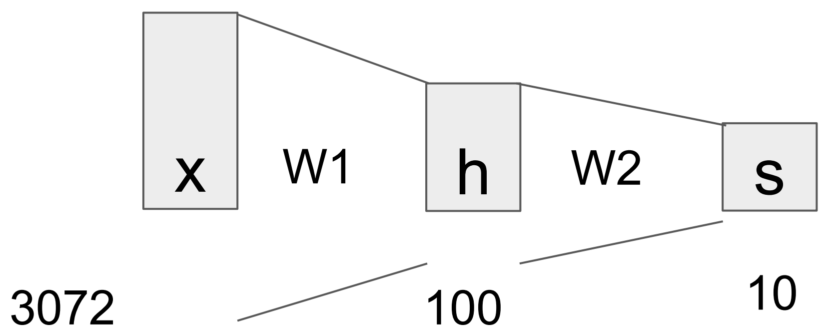
(**Before**) Linear score function: $f = Wx$

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

# Neural networks: without the brain stuff

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

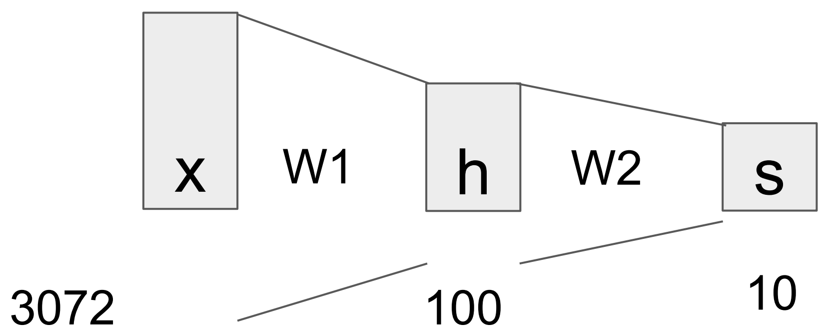
(**Now**) 2-layer Neural Network $$f = W_2 \max(0, W_1 x)$$

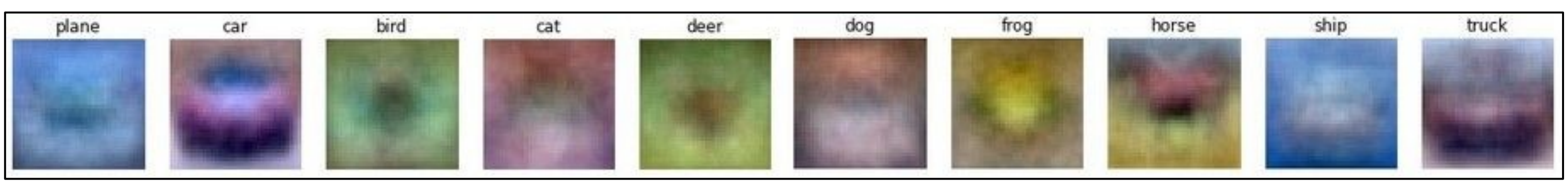


# Neural networks: without the brain stuff

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

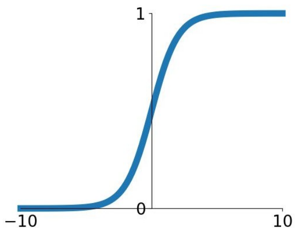
or 3-layer Neural Network

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

# Activation functions

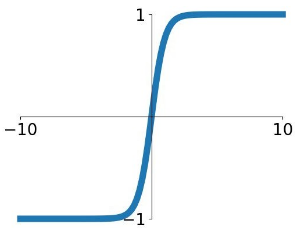
**Sigmoid**

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



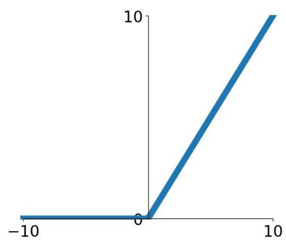

**tanh**

$\tanh(x)$



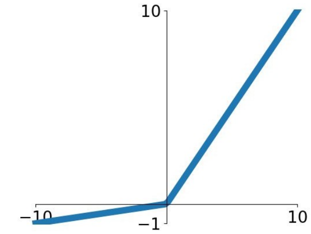

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

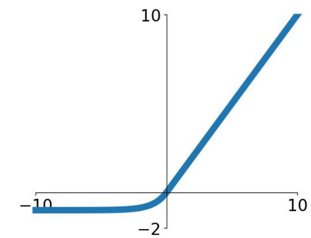



**Maxout**

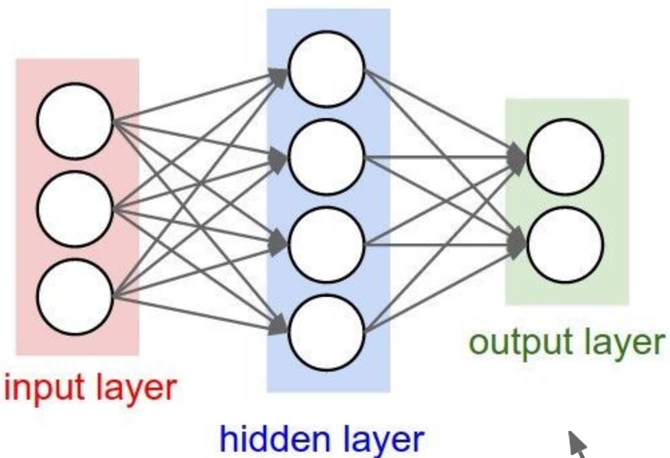
$\max(w_1^T x + b_1, w_2^T x + b_2)$

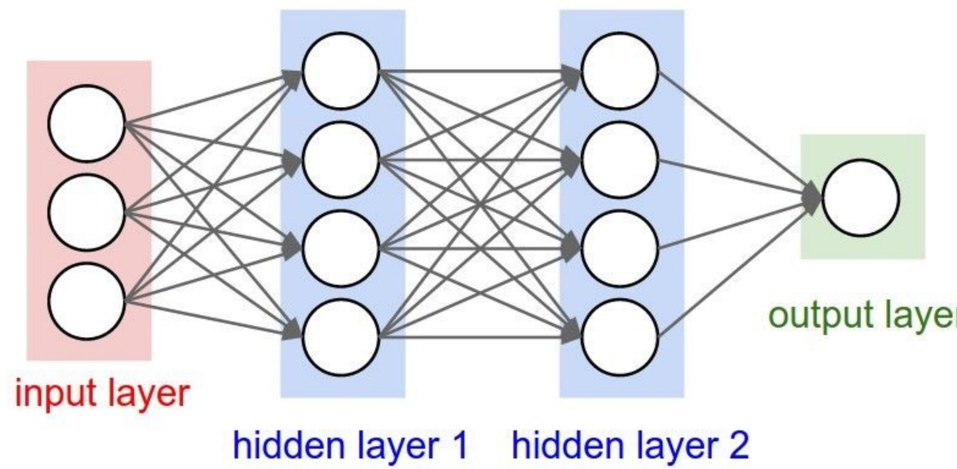
**ELU**

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



# Neural networks: Architectures




"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

**"Fully-connected" layers**

# Next: Convolutional Neural Networks

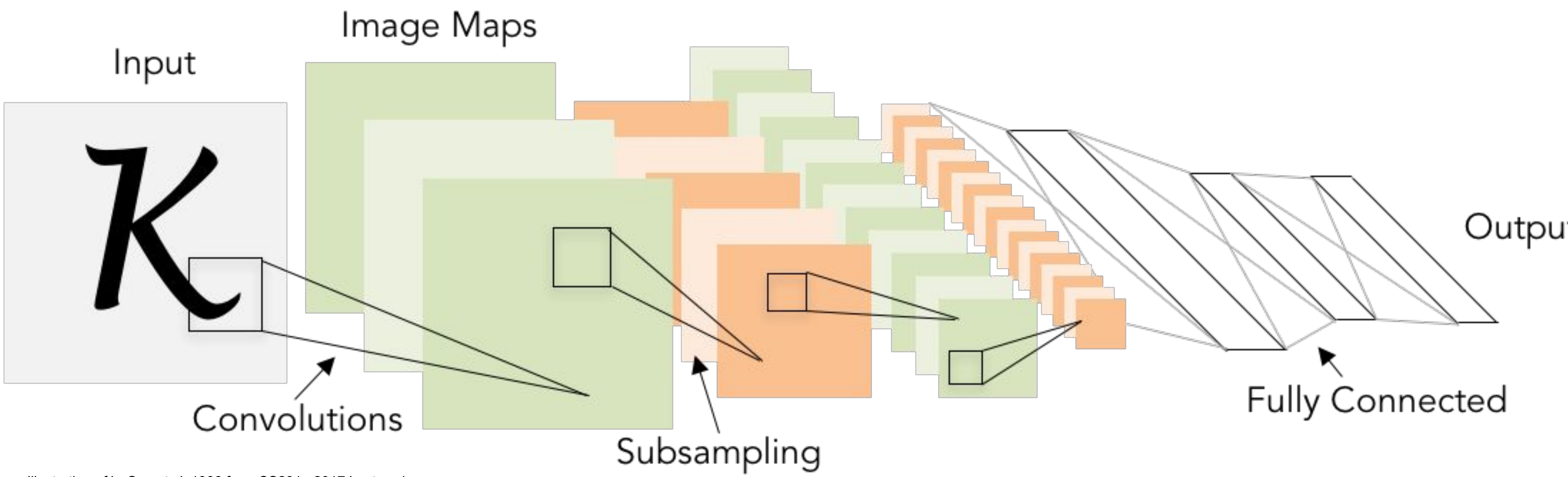
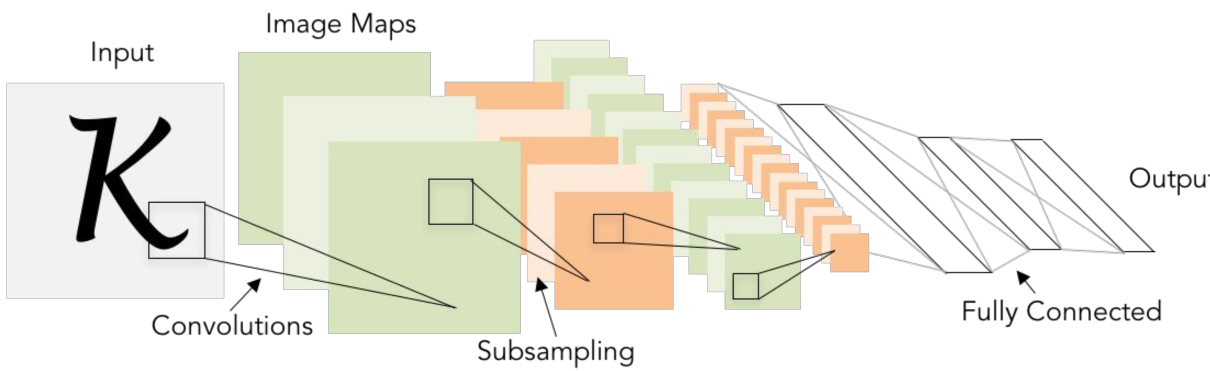



Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

**Gradient-based learning applied to document recognition**
*[LeCun, Bottou, Bengio, Haffner 1998]*

# A bit of history:




LeNet-5

# A bit of history:

## **ImageNet Classification with Deep Convolutional Neural Networks**

*[Krizhevsky, Sutskever, Hinton, 2012]*

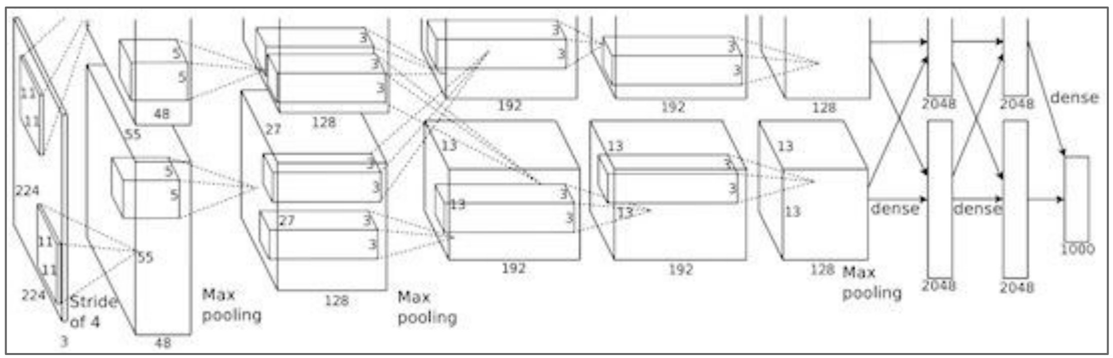



Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

# Fast-forward to today: ConvNets are everywhere




self-driving cars



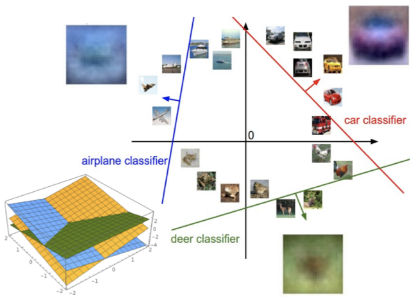

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

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

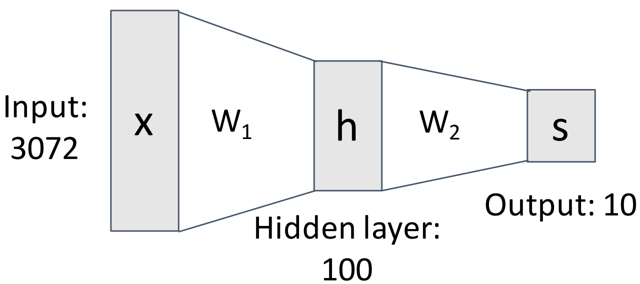
# Convolutional Neural Networks

(First without the brain stuff)

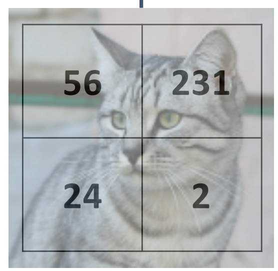
$f(x,W) = Wx$




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

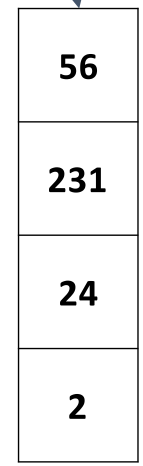



Stretch pixels into column



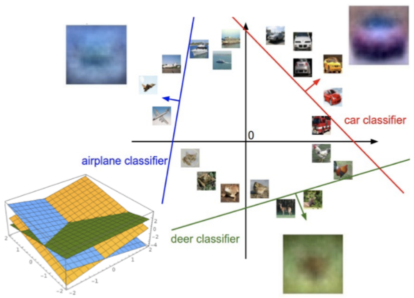

Input image
(2, 2)

**Problem**: So far our classifiers don't respect the spatial structure of images!

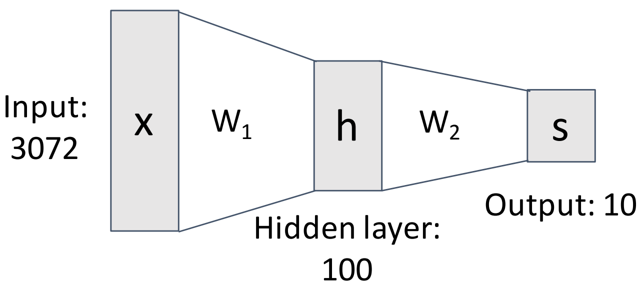



(4,)

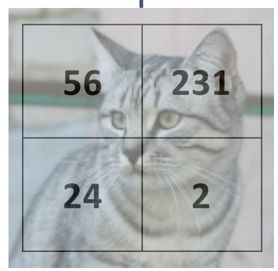
f(x,W) = Wx




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



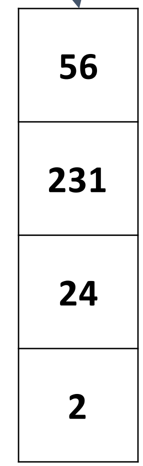

Stretch pixels into column




Input image
(2, 2)

**Problem**: So far our classifiers don’t respect the spatial structure of images!

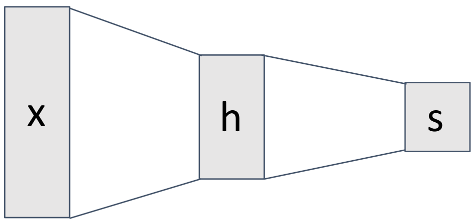
**Solution**: Define new computational nodes that operate on images!



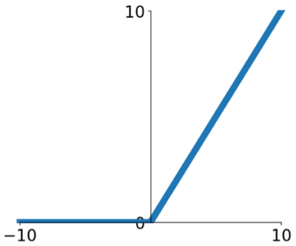

(4,)

# Components of a Fully-Connected Network

Fully-Connected Layers

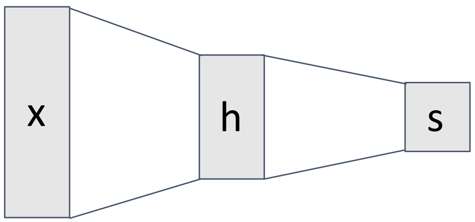



Activation Function

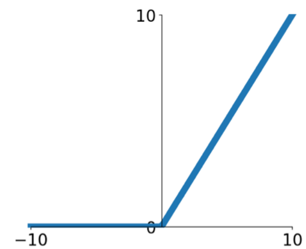


# Components of a Convolutional Network

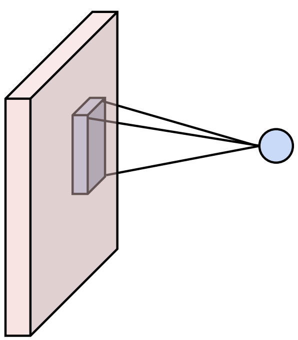
Fully-Connected Layers



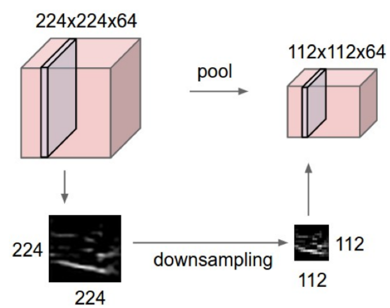

Activation Function




Convolution Layers



Pooling Layers

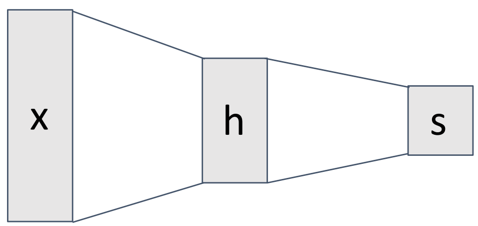



Normalization

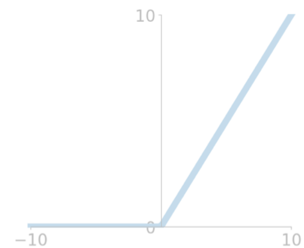
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Components of a Convolutional Network

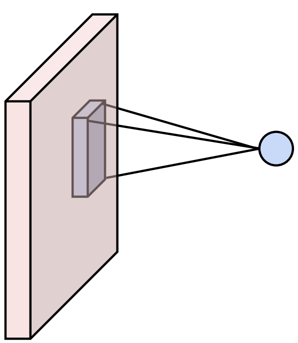
## Fully-Connected Layers



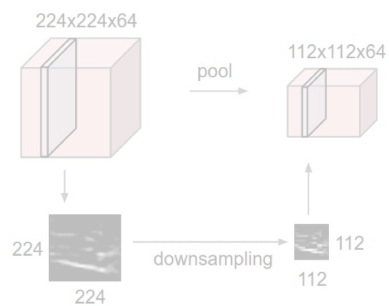

## Activation Function




## Convolution Layers



## Pooling Layers

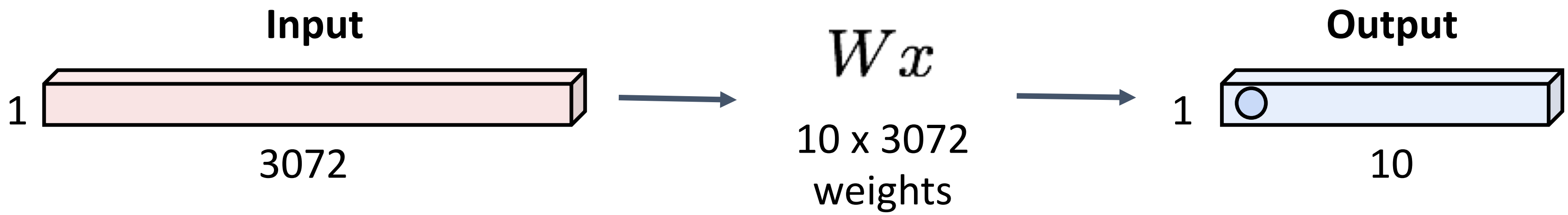



## Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

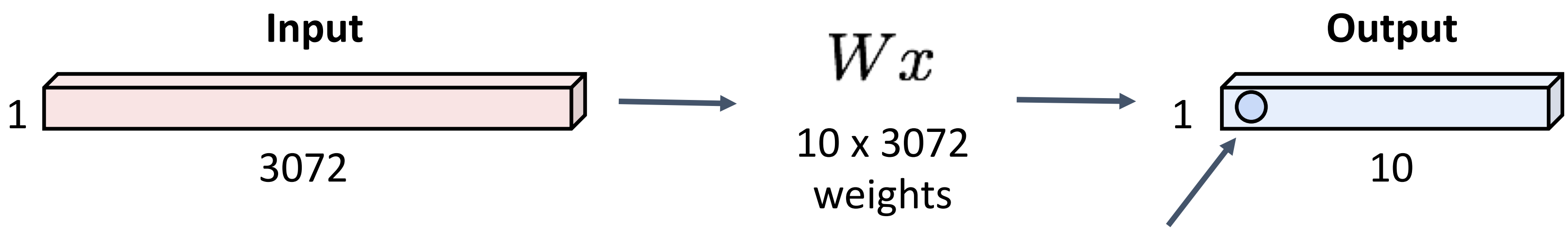
# Fully-Connected Layer

32x32x3 image -> stretch to 3072 x 1



# Fully-Connected Layer

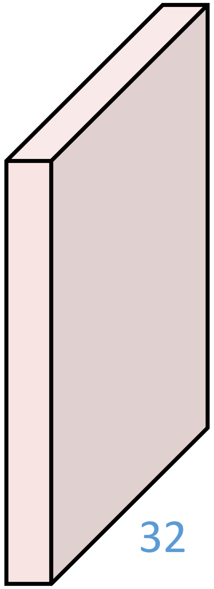
32x32x3 image -> stretch to 3072 x 1




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

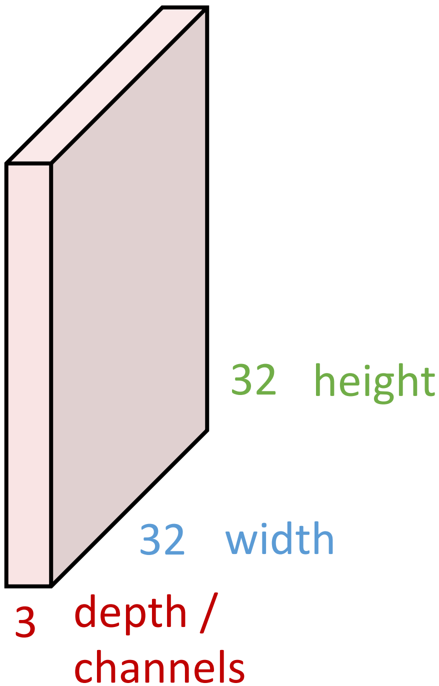
# Convolution Layer

3x32x32 image: preserve spatial structure

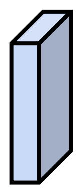


# Convolution Layer

3x32x32 image




3x5x5 filter

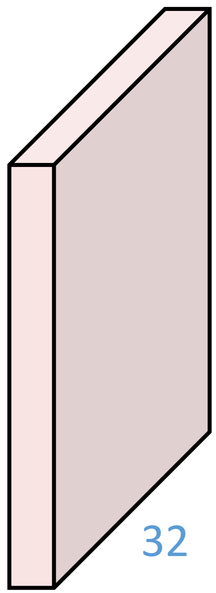


**Convolve** the filter with the image
i.e. "slide over the image spatially,
computing dot products"

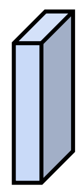
# Convolution Layer

Filters (almost) always extend the full depth of the input volume

3x32x32 image

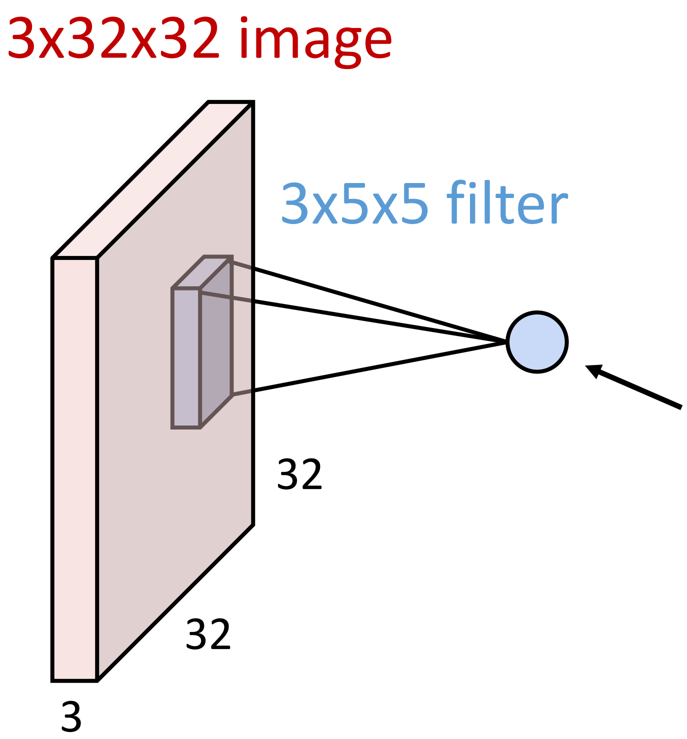



3x5x5 filter



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

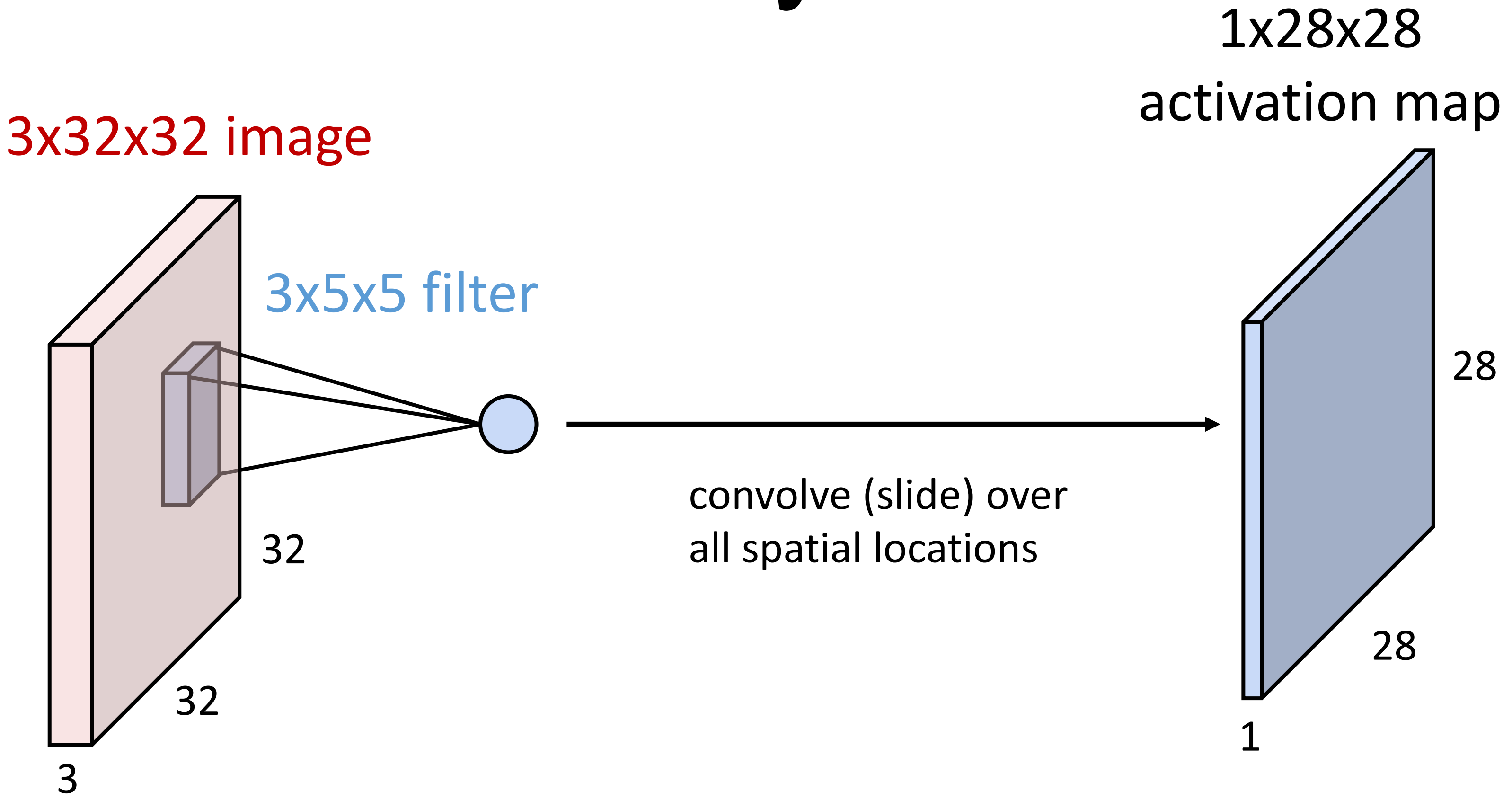
# Convolution Layer



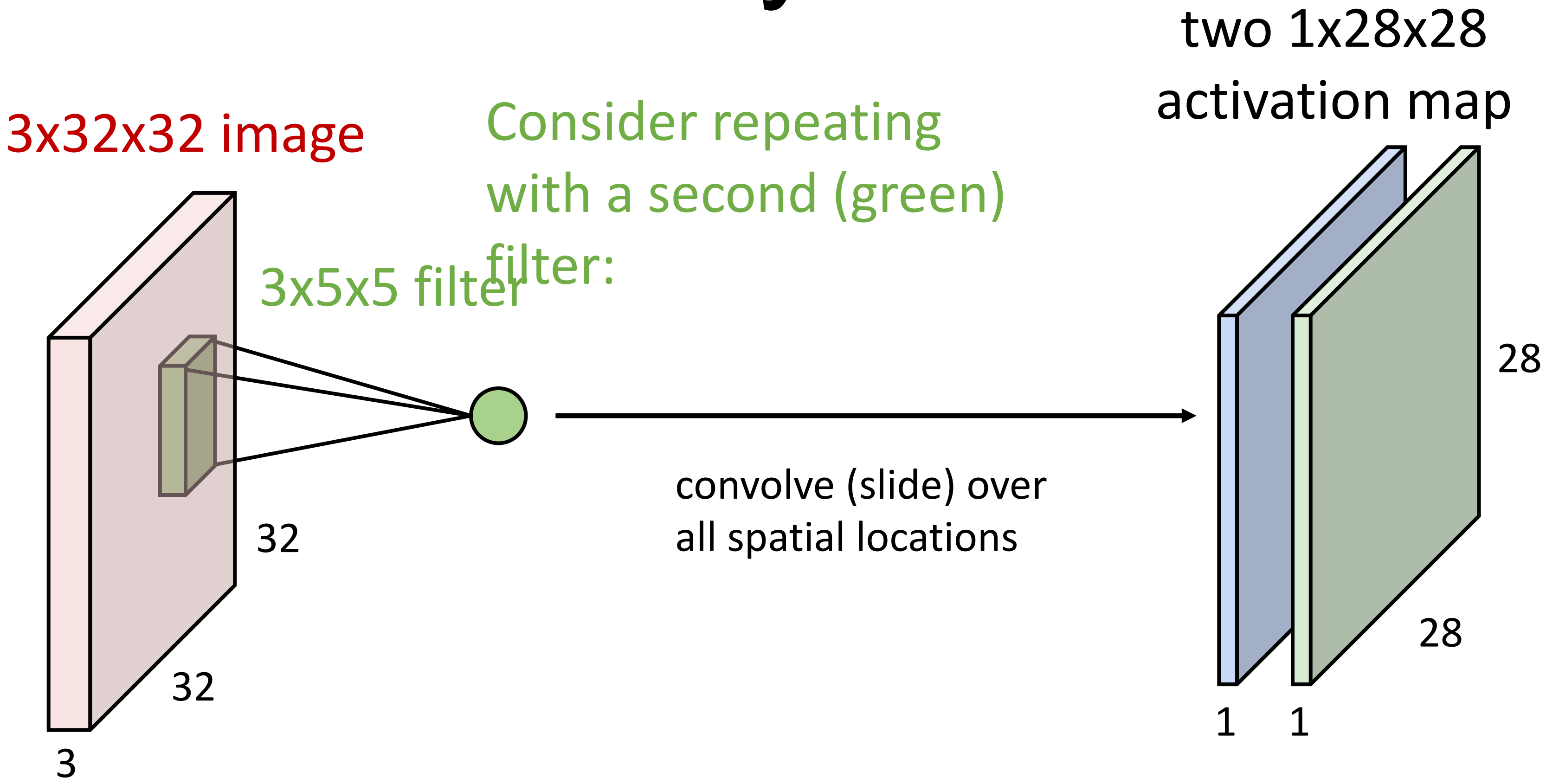

**1 number:**
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

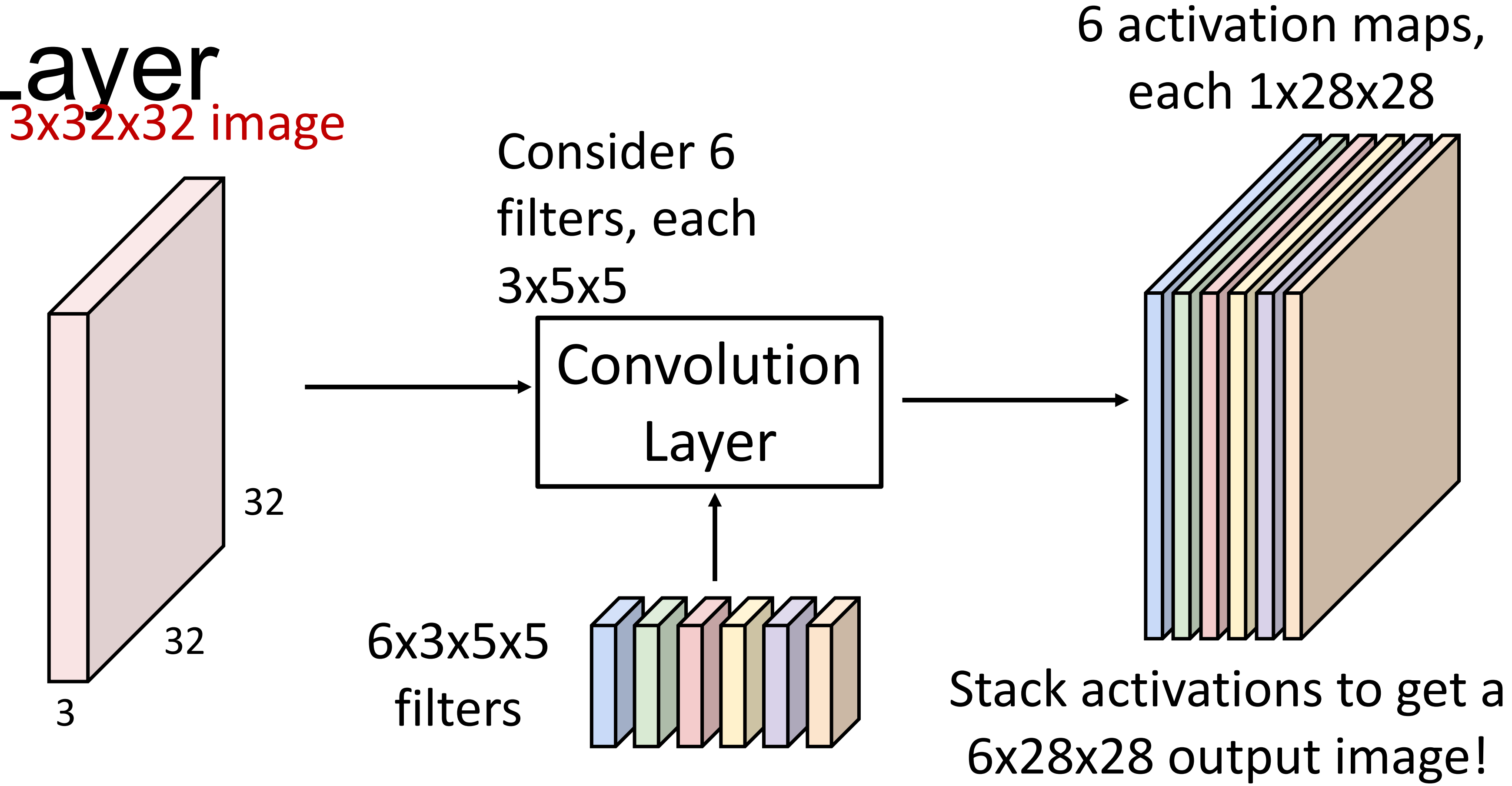
# Convolution Layer



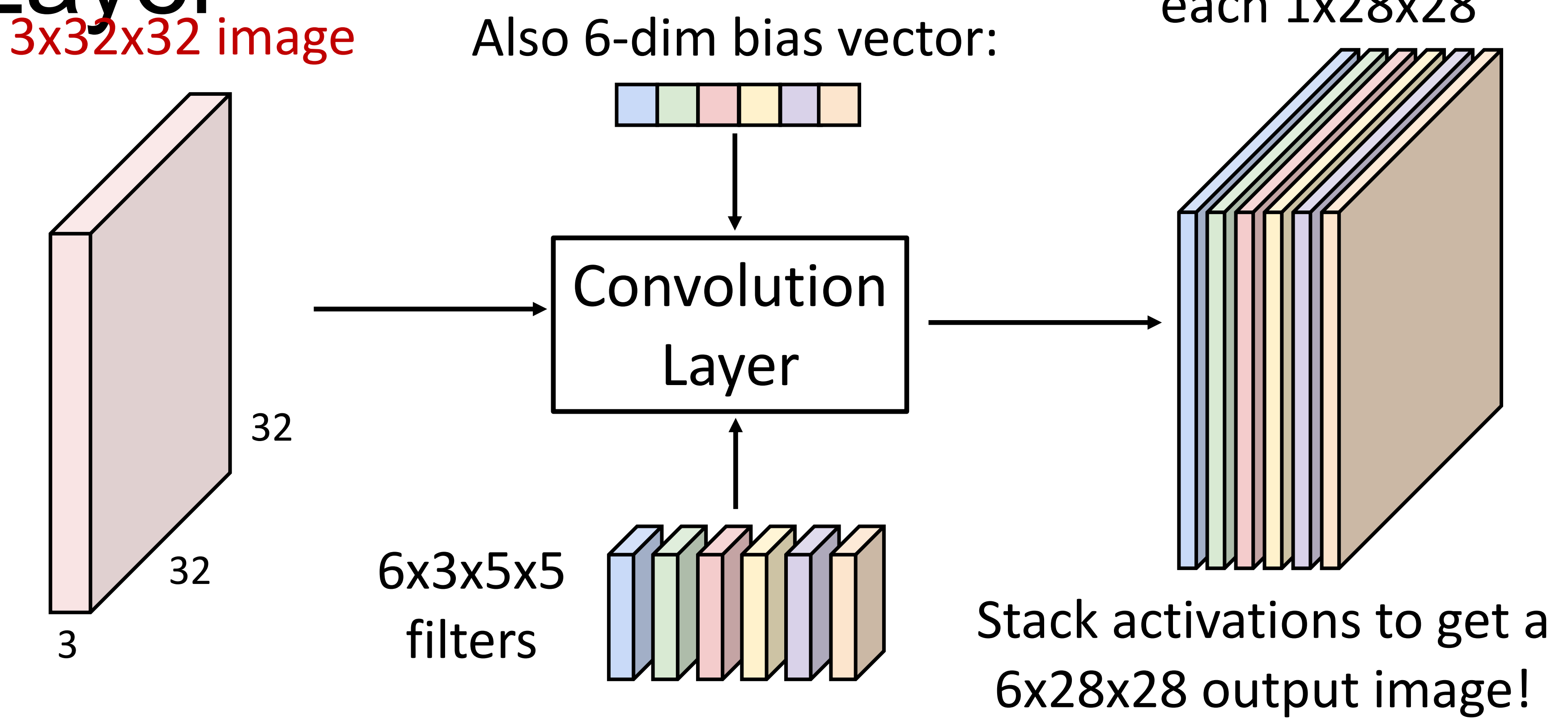
# Convolution Layer



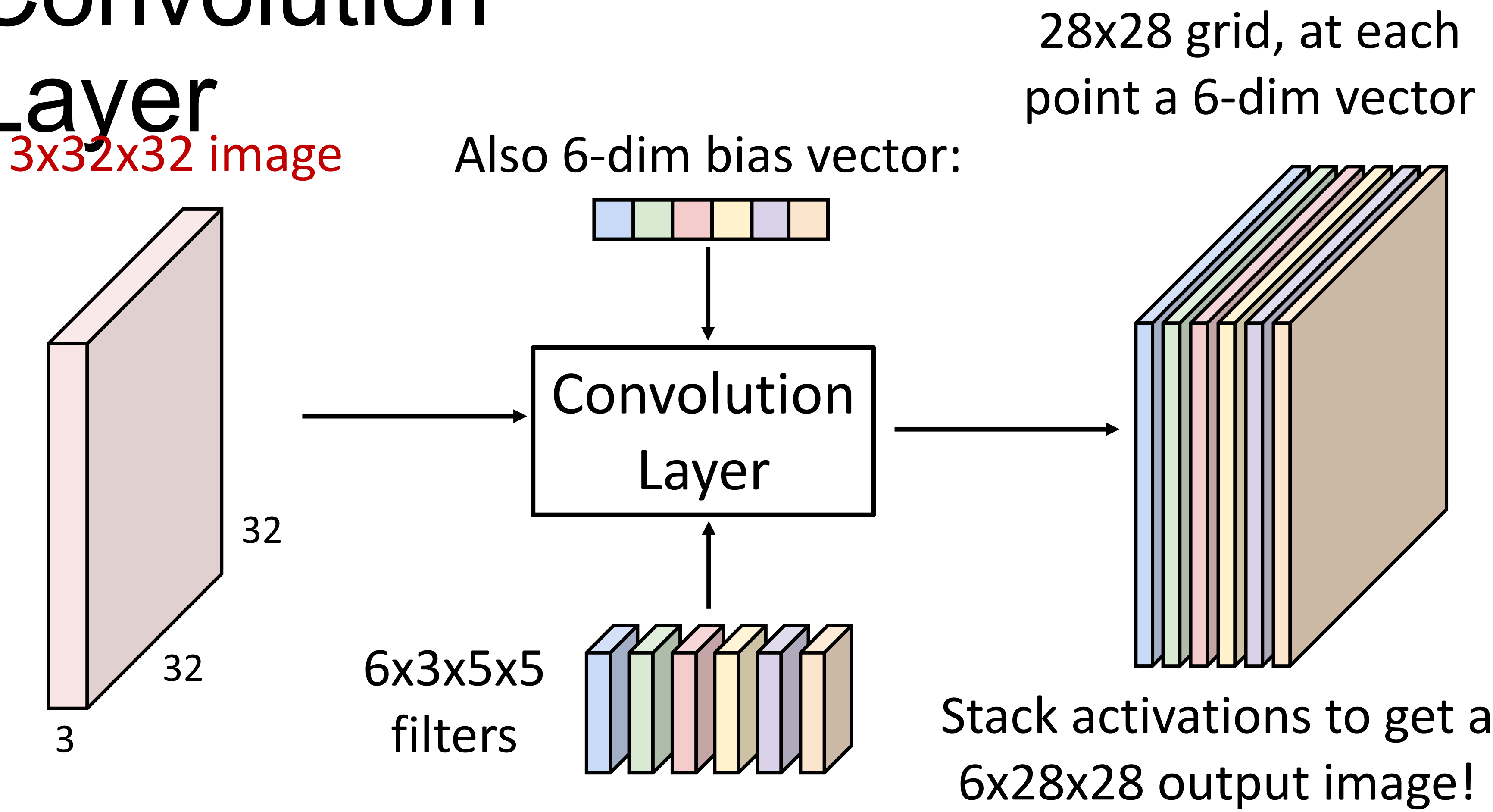
# Convolution Layer



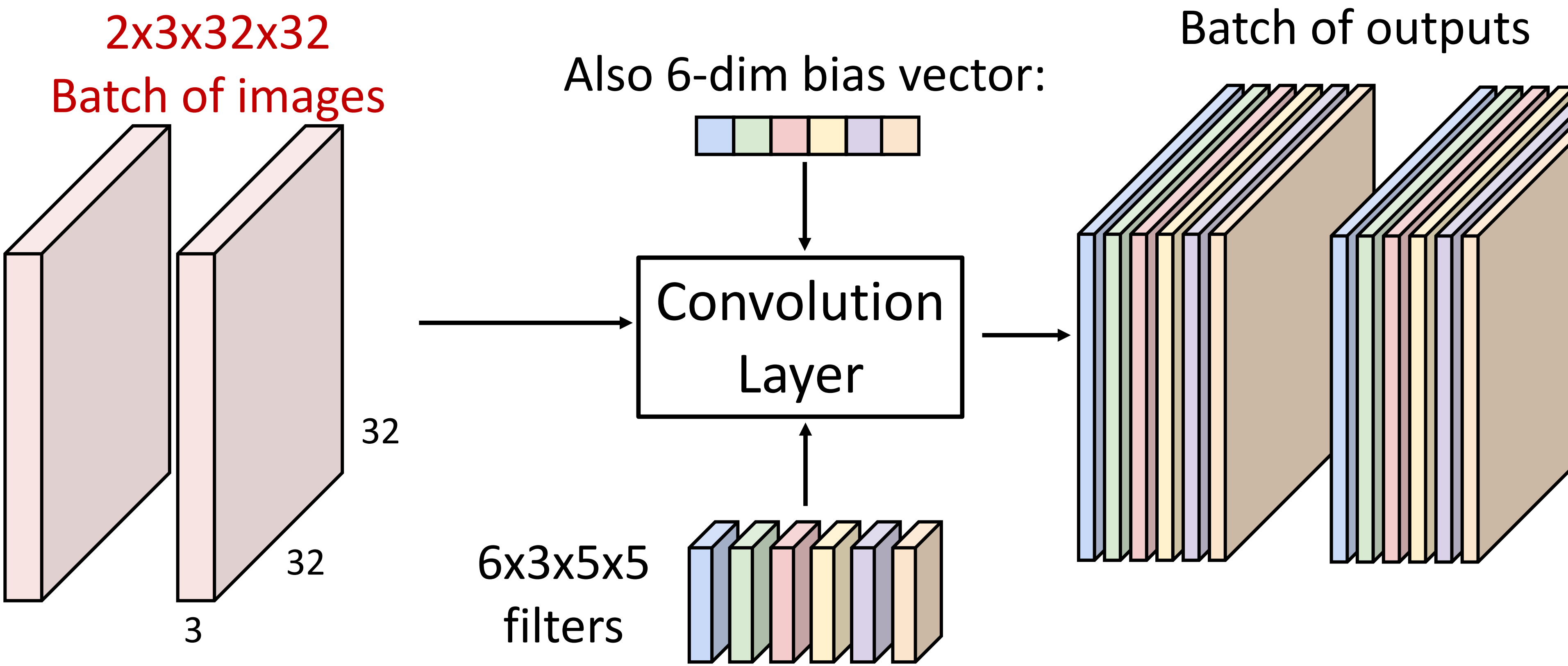
# Convolution Layer



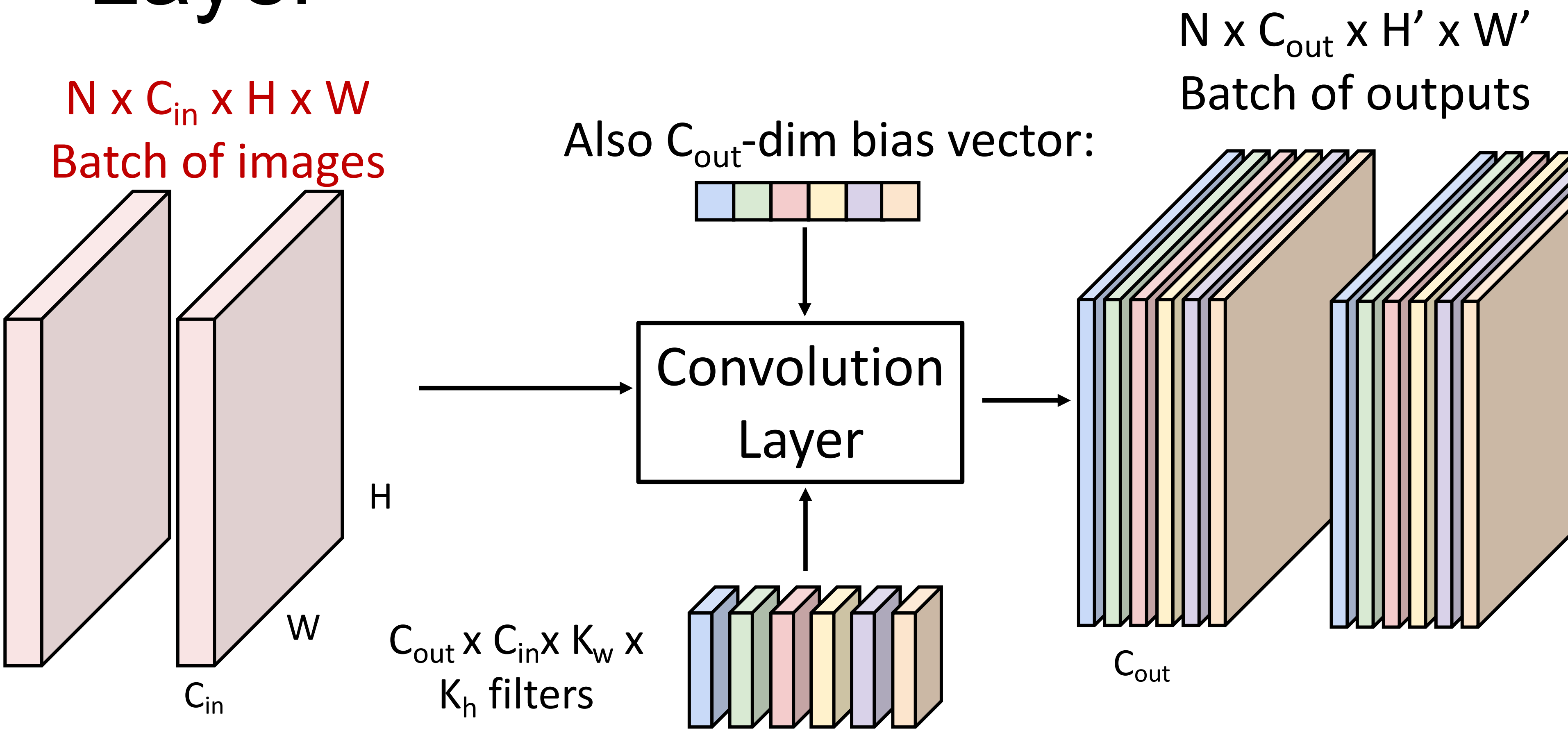
# Convolution Layer



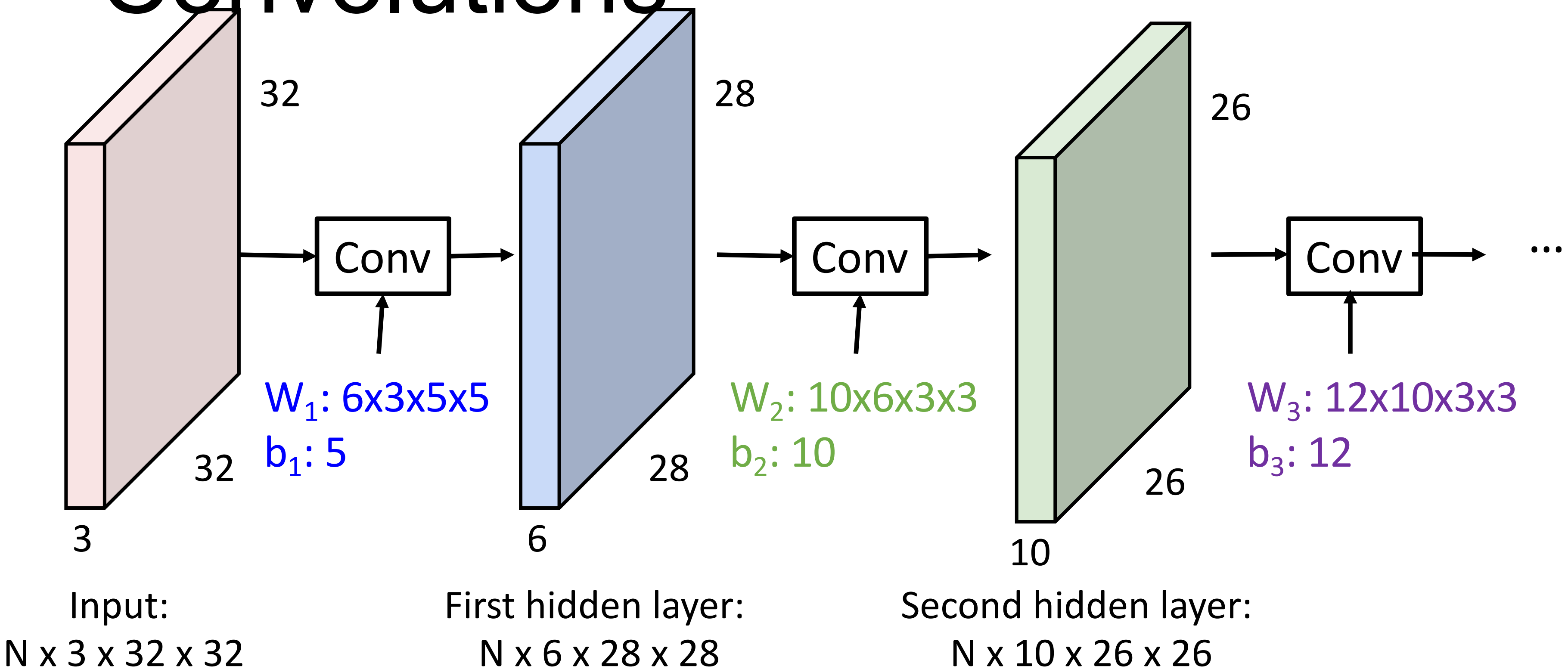
# Convolution Layer



# Convolution Layer



# Stacking Convolutions



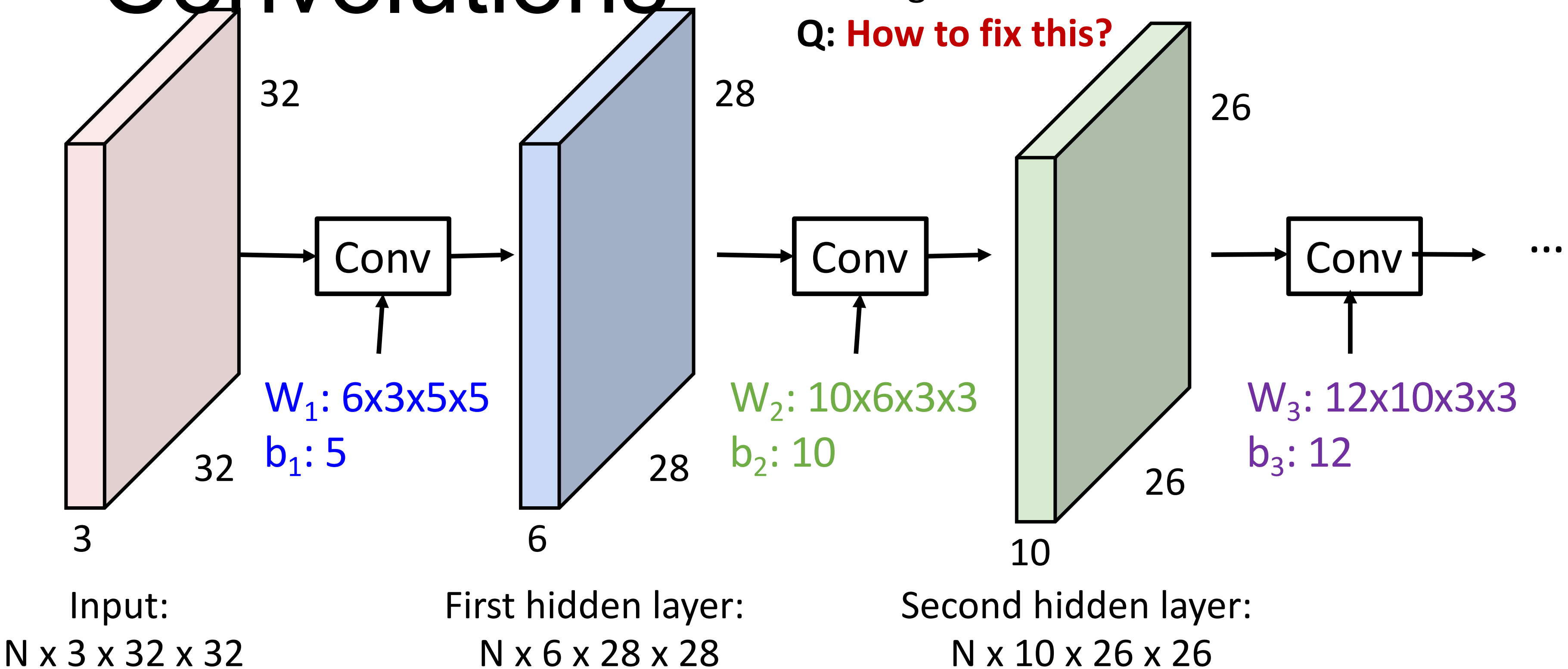
# Stacking Convolutions

(Recall $y=W_2W_1x$ is a linear classifier)

**Q**: What happens if we stack two convolution layers?

**A**: We get another convolution!

**Q: How to fix this?**



# Stacking Convolutions

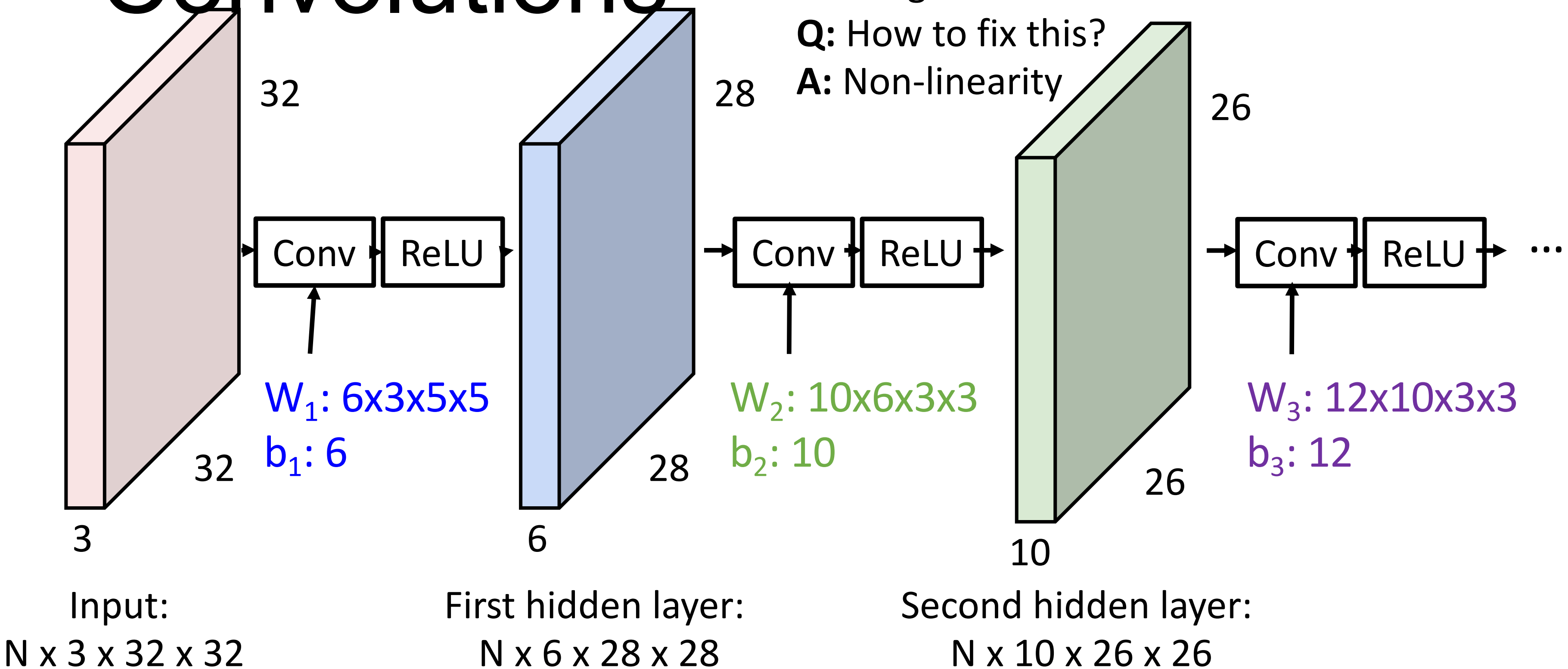
**Q**: What happens if we stack two convolution layers?

**A**: We get another convolution!

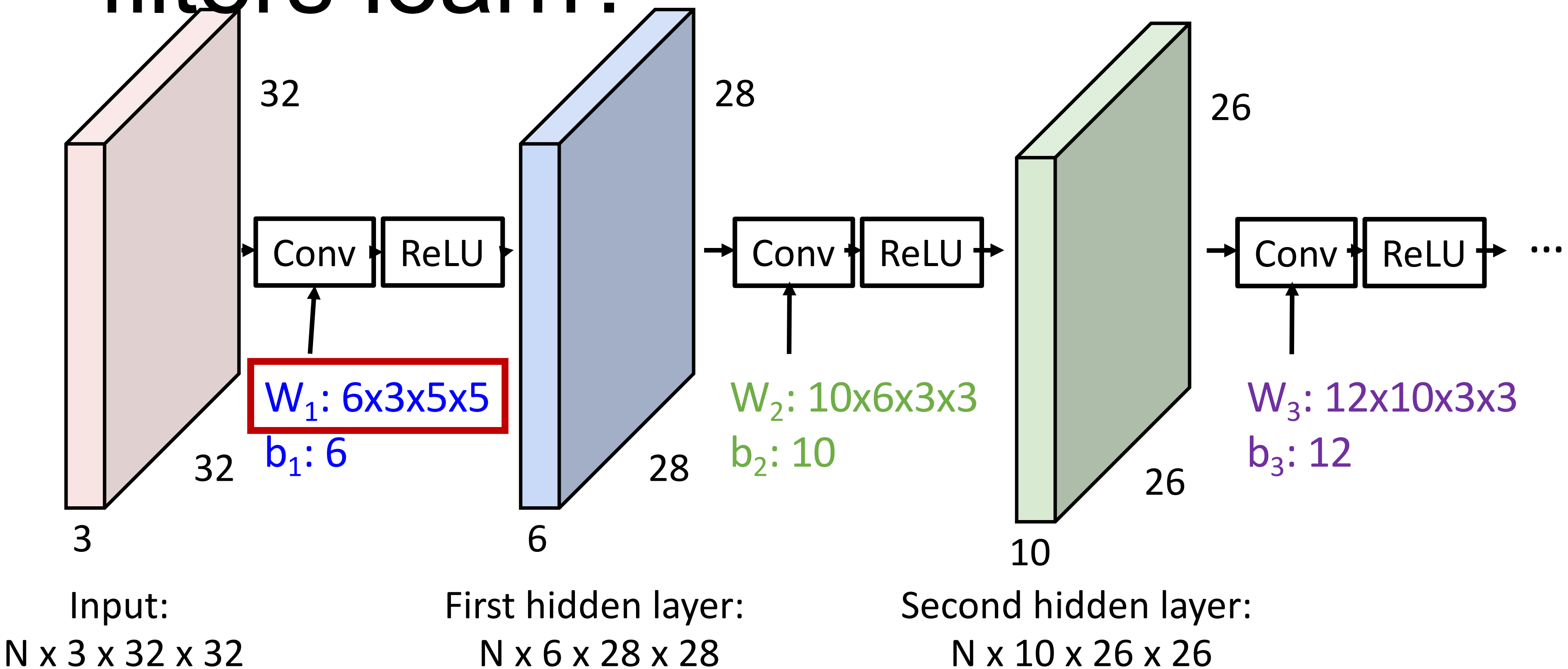
(Recall $y=W_2W_1x$ is a linear classifier)

**Q:** How to fix this?

**A:** Non-linearity



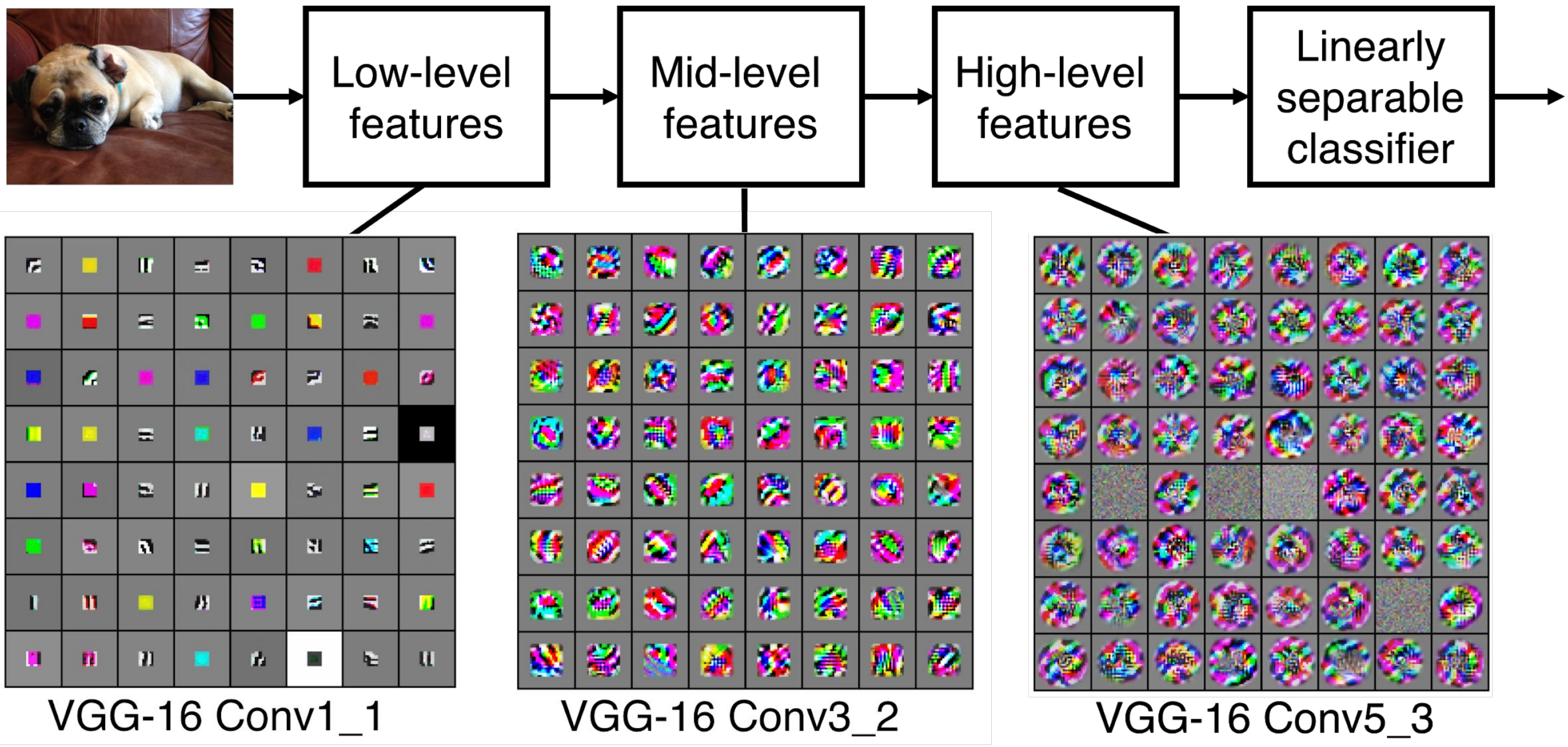
# What do convolutional filters learn?



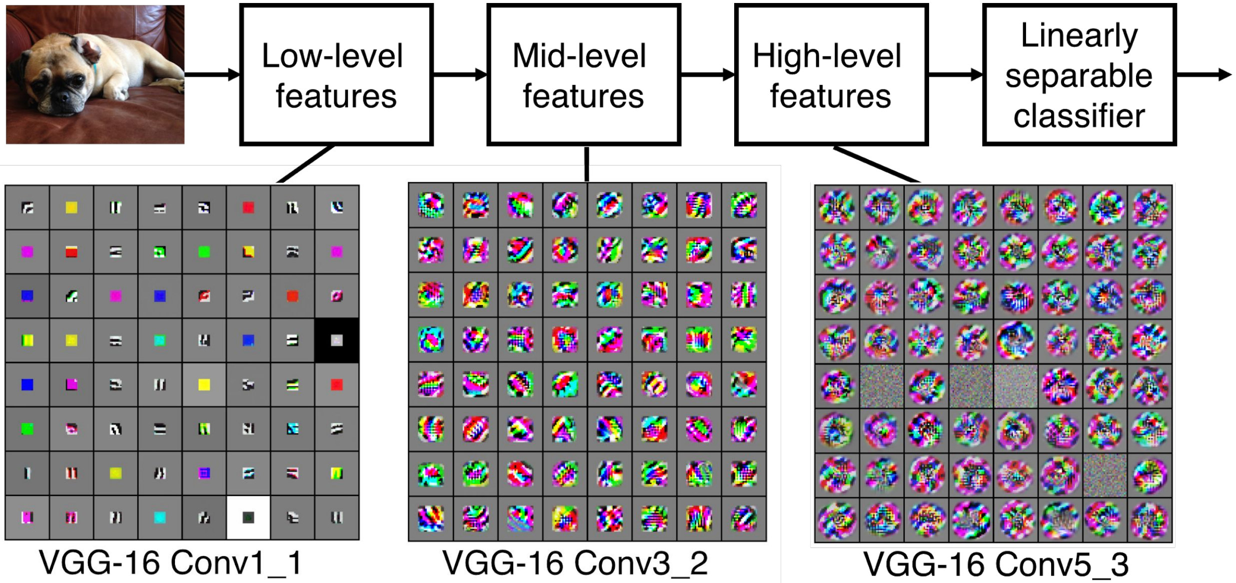
# Preview

[Zeiler and Fergus 2013]

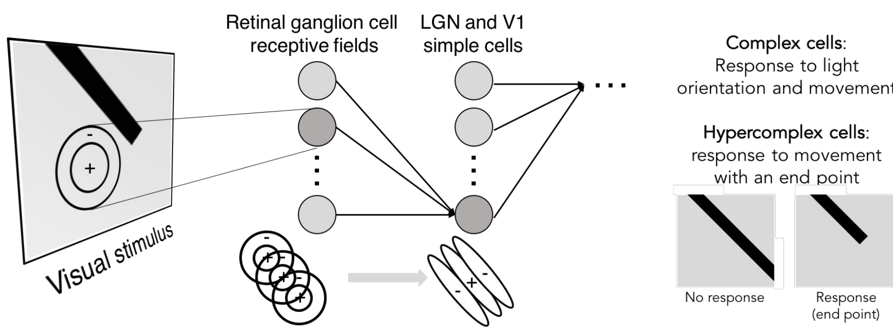
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



# Preview

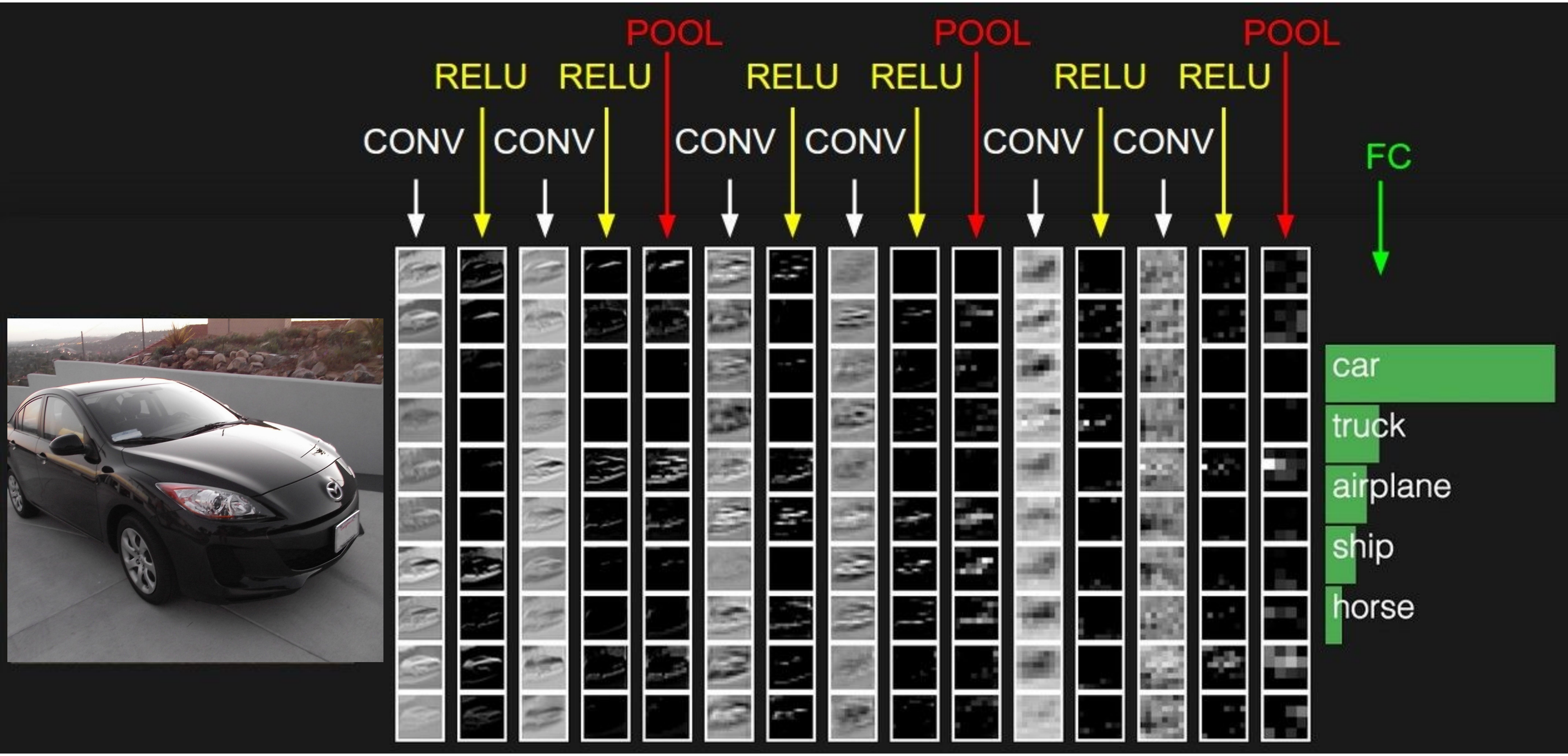


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3

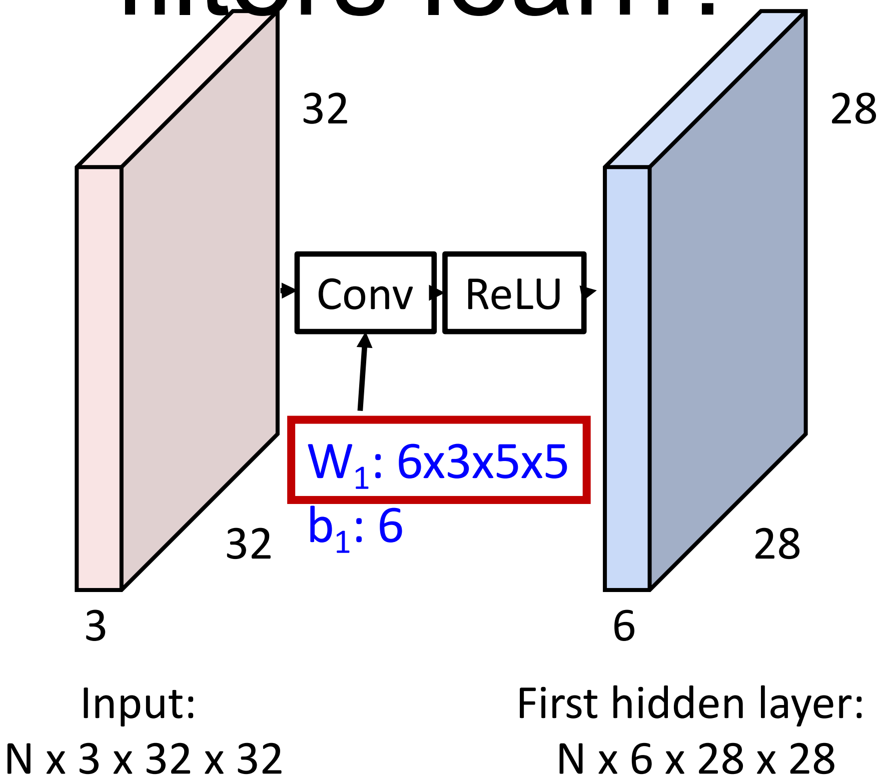



Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells: Response to light orientation and movement
Hypercomplex cells: response to movement with an end point
No response
Response (end point)

preview:

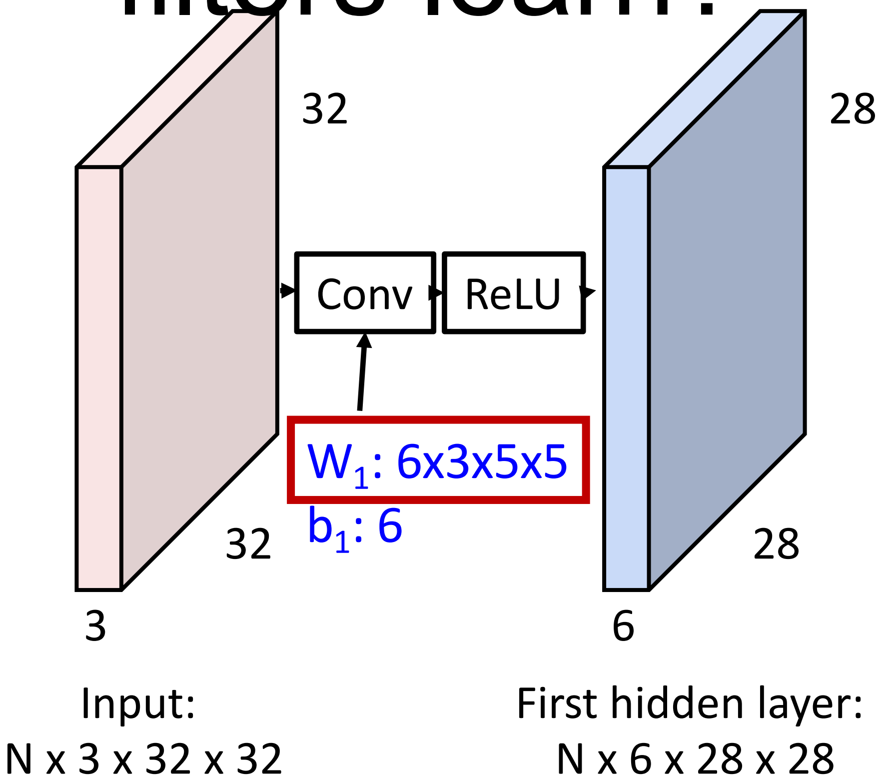

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

# What do convolutional filters learn?

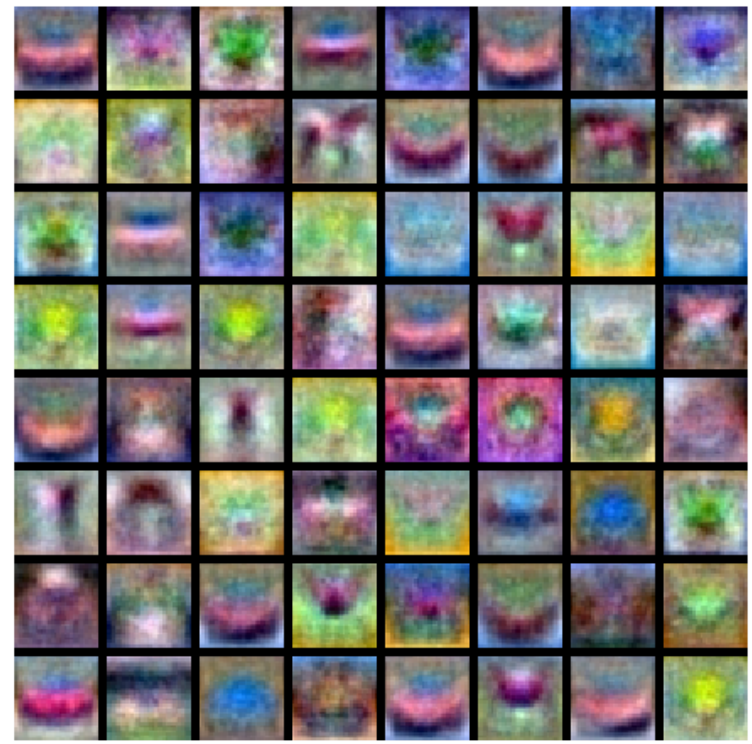



Linear classifier: One template per class

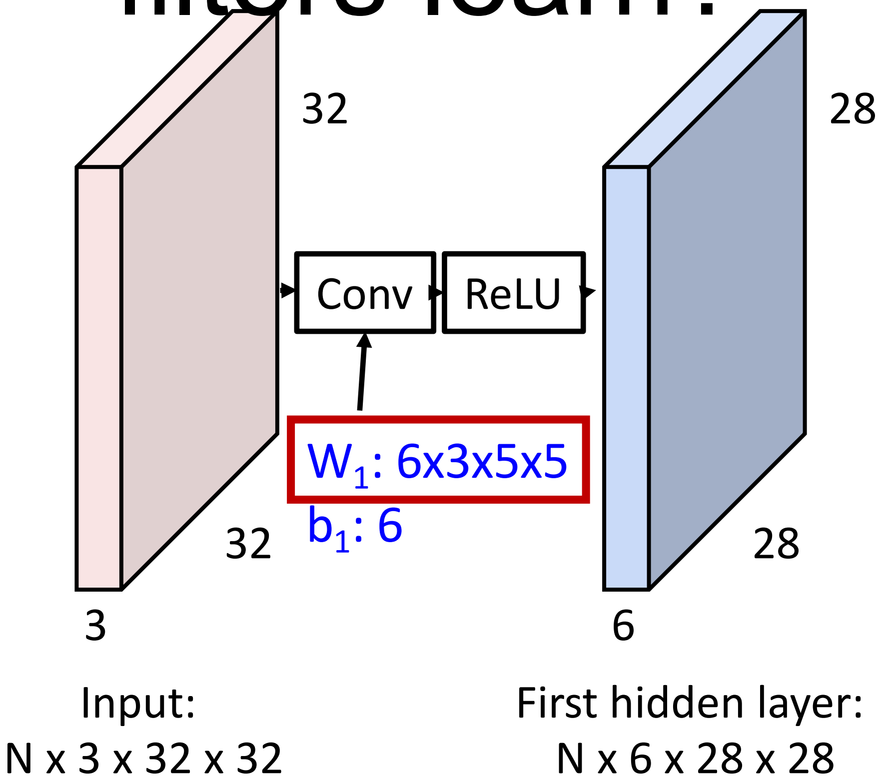
# What do convolutional filters learn?



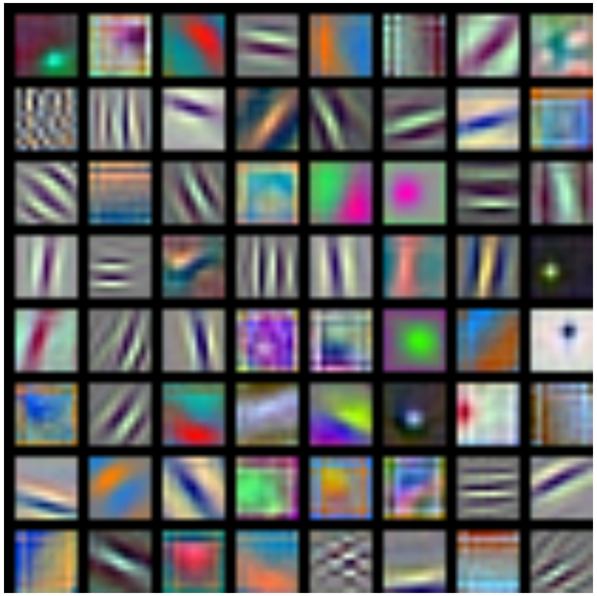

MLP: Bank of whole-image templates



# What do convolutional filters learn?

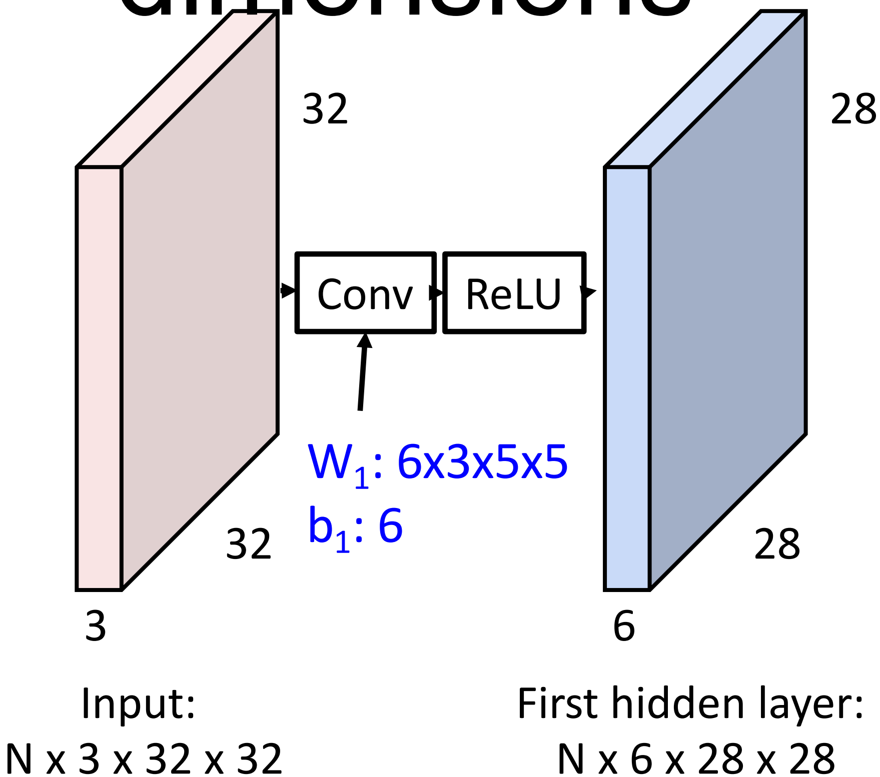



First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

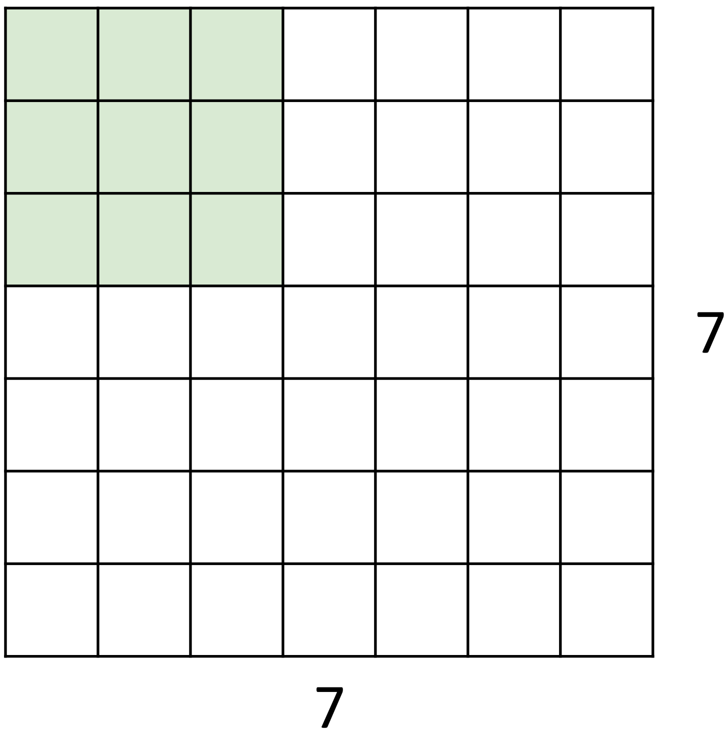


AlexNet: 64 filters, each 3x11x11

# A closer look at spatial dimensions



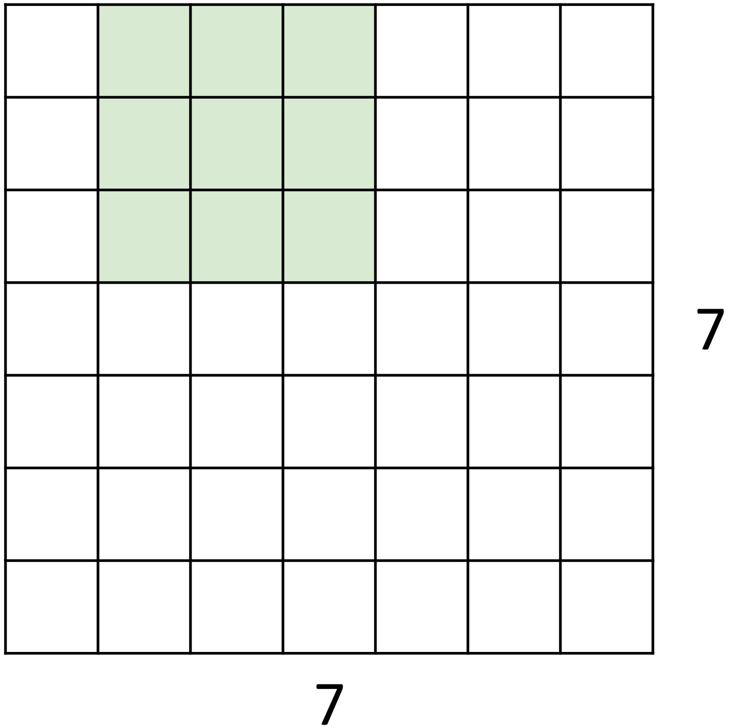
# A closer look at spatial dimensions




Input: 7x7

Filter: 3x3

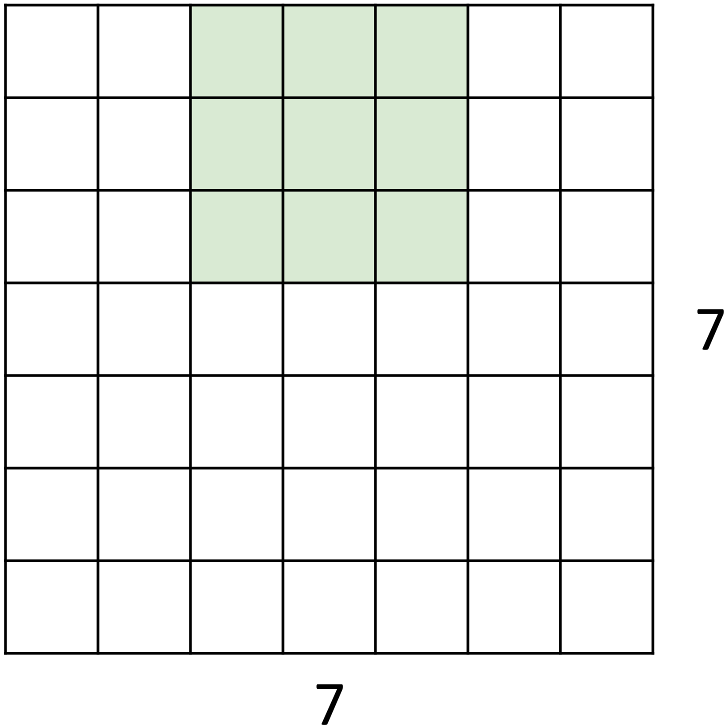
# A closer look at spatial dimensions




Input: 7x7

Filter: 3x3

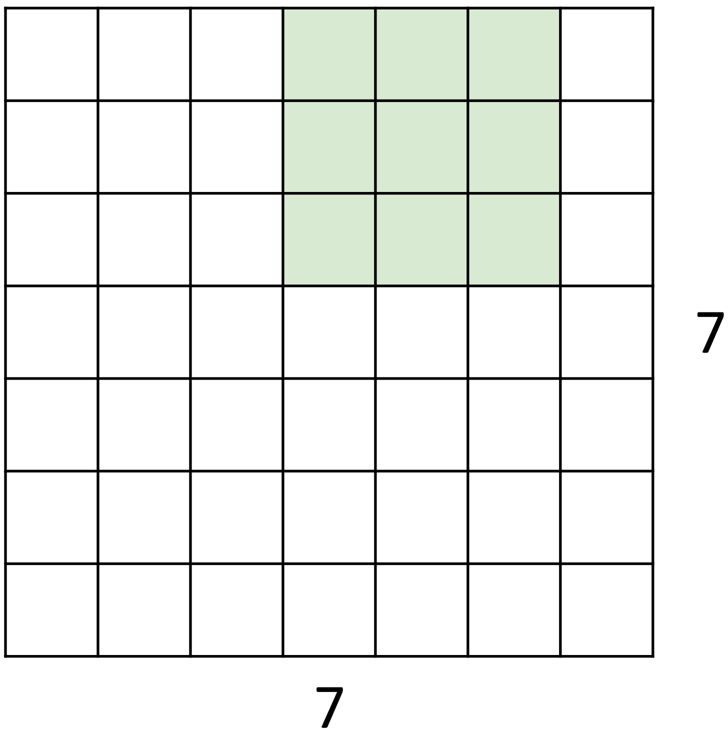
# A closer look at spatial dimensions




Input: 7x7

Filter: 3x3

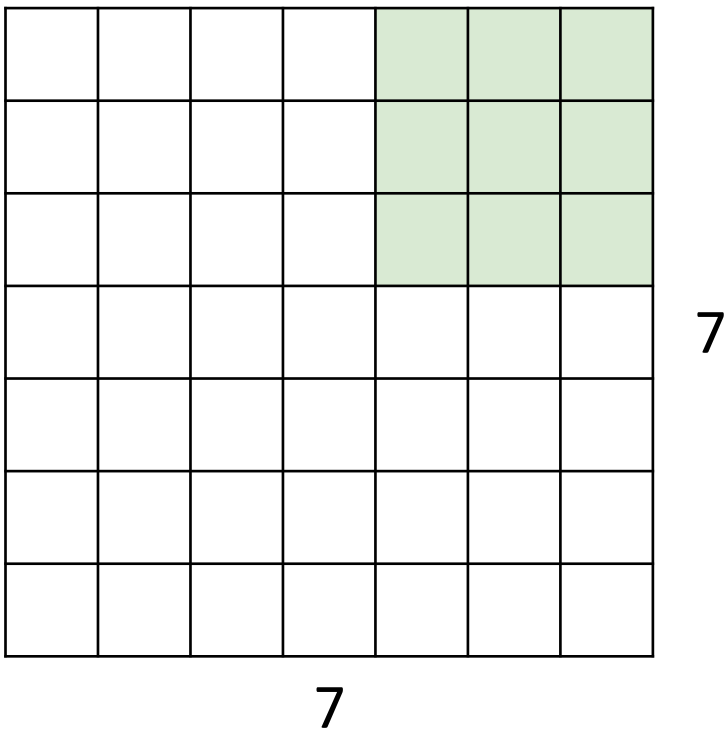
# A closer look at spatial dimensions




Input: 7x7

Filter: 3x3

# A closer look at spatial dimensions

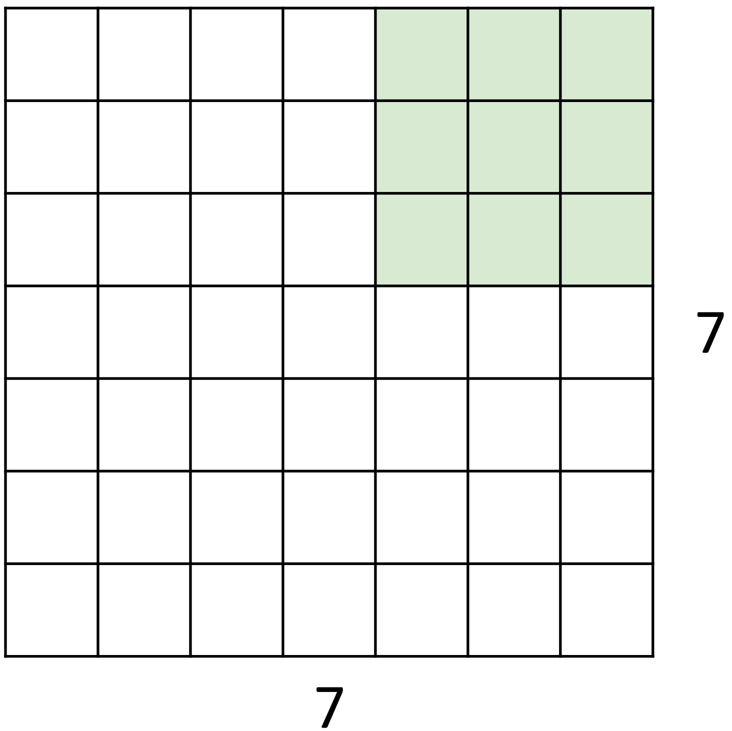



Input: 7x7

Filter: 3x3

Output: 5x5

# A closer look at spatial dimensions

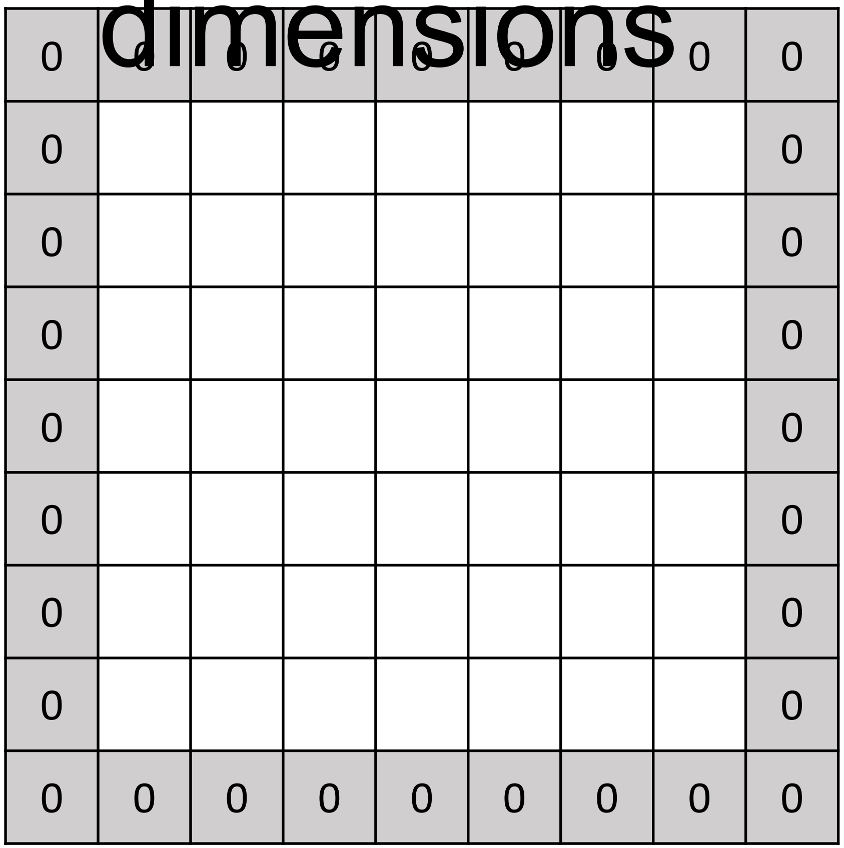



Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem:
Feature maps "shrink" with each layer!

# A closer look at spatial dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem: Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

# A closer look at spatial dimensions

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | | | | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

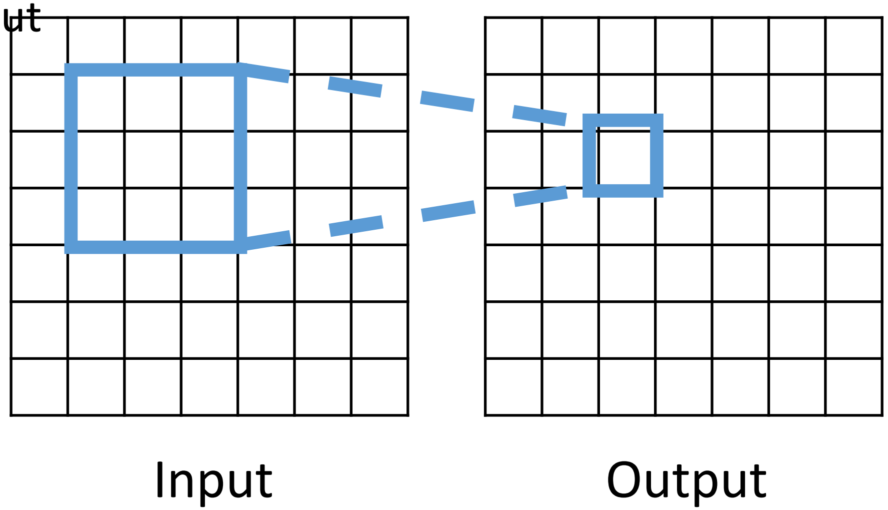
Padding: P

Output: W – K + 1 + 2P

Very common:
Set P = (K – 1) / 2 to make output have same size as input!

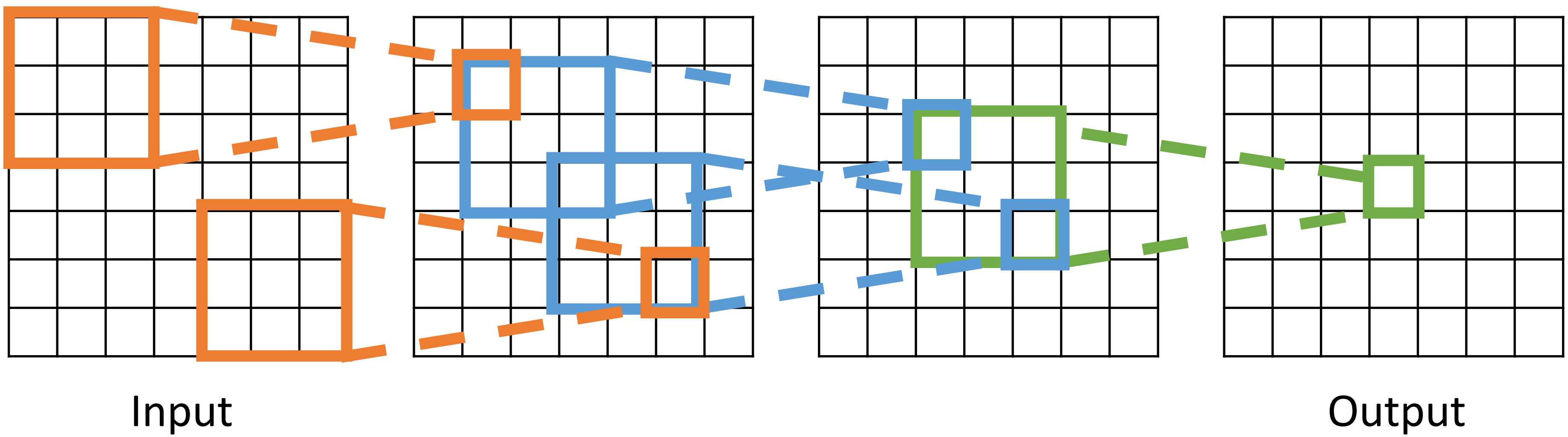
# Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



# Receptive Fields

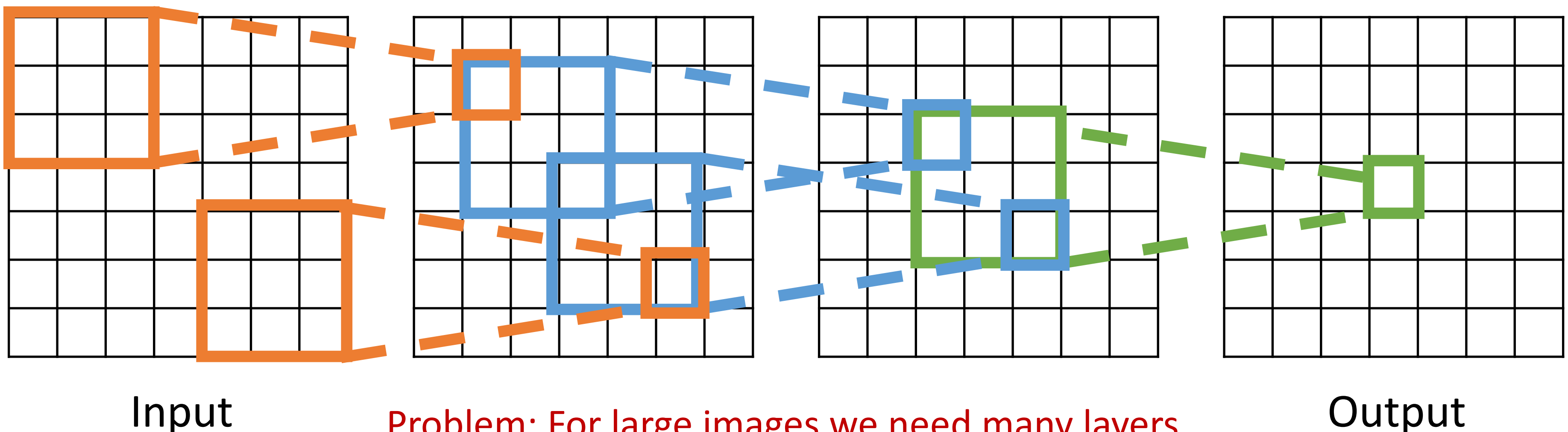
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)




Be careful – "receptive field in the input" vs "receptive field in the previous layer"
Hopefully clear from context!

# Receptive Fields

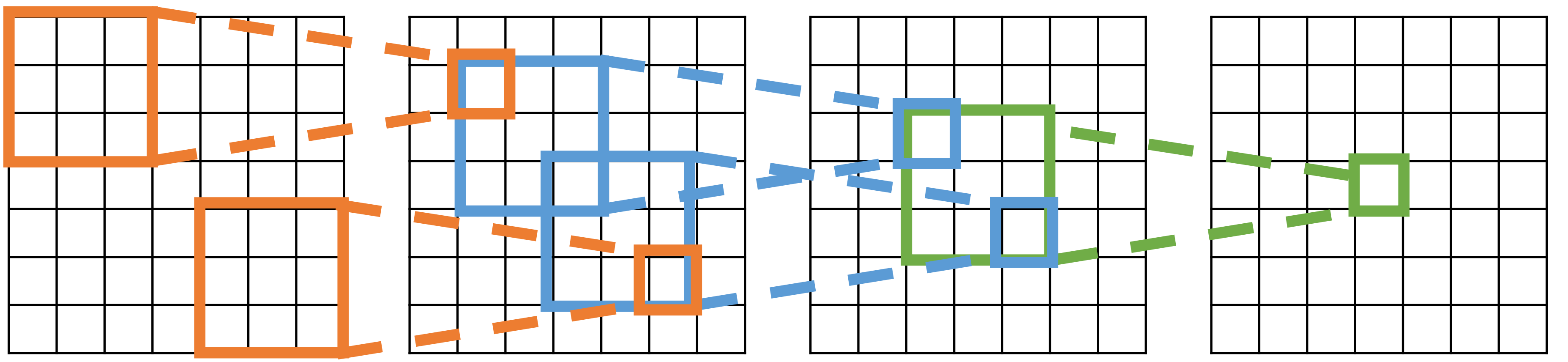
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)




Problem: For large images we need many layers for each output to "see" the whole image image

# Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



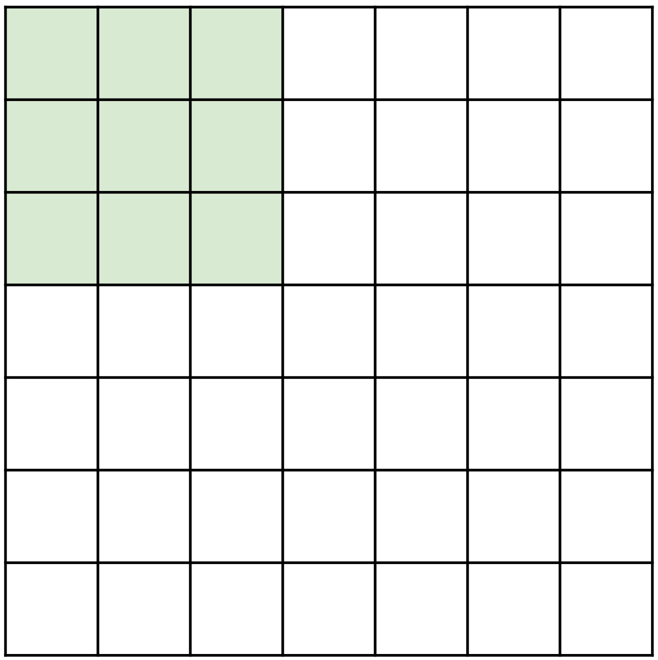
Input

Output

Problem: For large images we need many layers for each output to “see” the whole image image

Solution: Downsample inside the network

# Strided Convolution

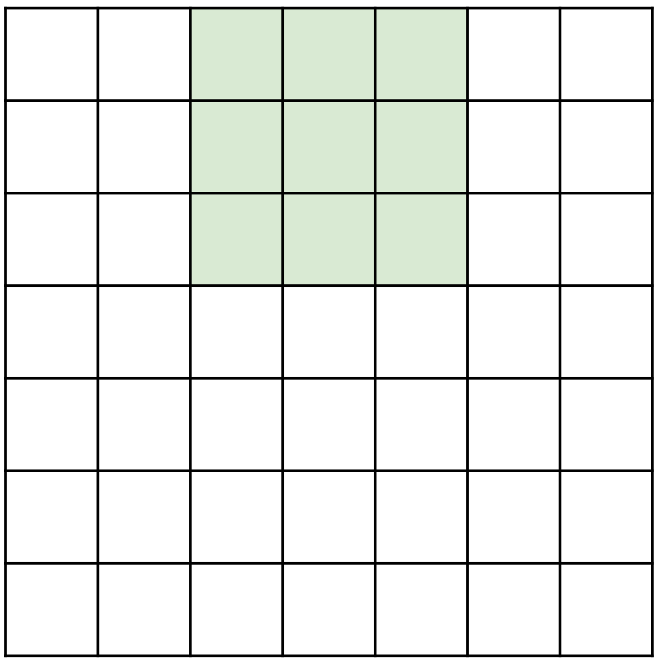


Input: 7x7

Filter: 3x3

Stride: 2

# Strided Convolution

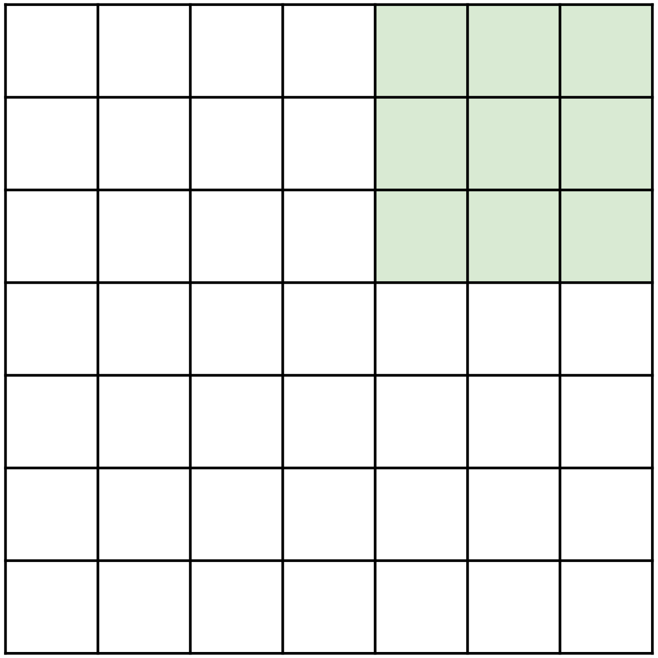


Input: 7x7

Filter: 3x3

Stride: 2

# Strided Convolution



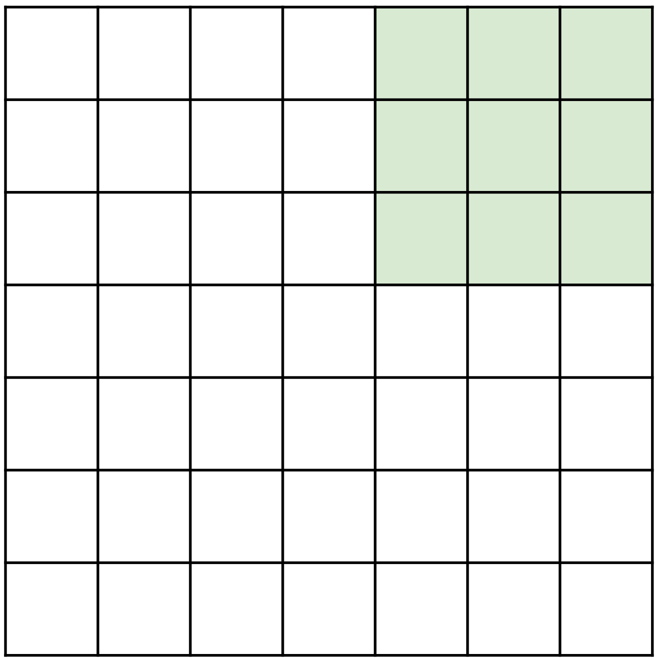
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

# Strided Convolution



Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

In general:

Input: W

Filter: K

Padding: P

Stride: S

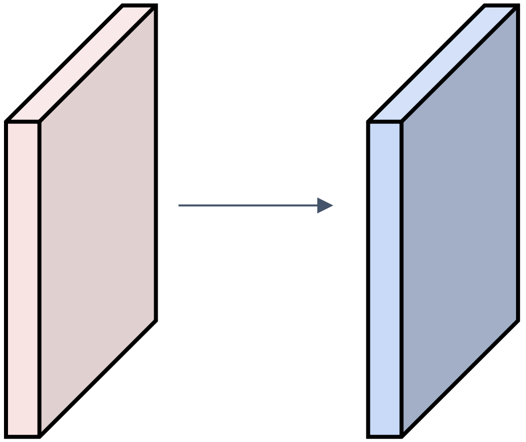
Output: (W – K + 2P) / S + 1

# Convolution Example

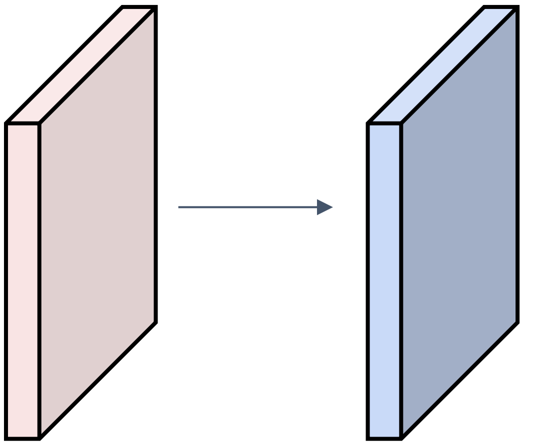
Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

Output volume size: ?



# Convolution Example



Input volume: 3 x **32** x **32**

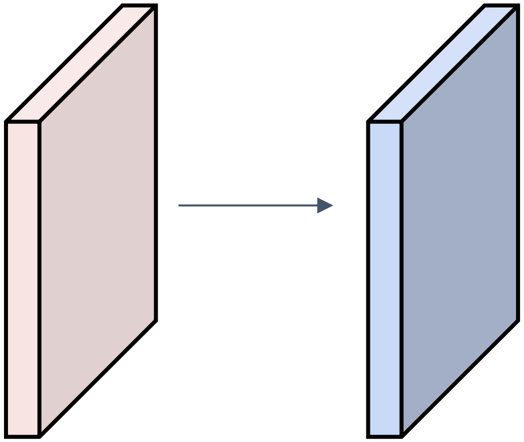
**10 5x5** filters with stride **1**, pad **2**

Output volume size:

(**32**+2***2**-**5**)/**1**+1 = 32 spatially, so

10 x 32 x 32

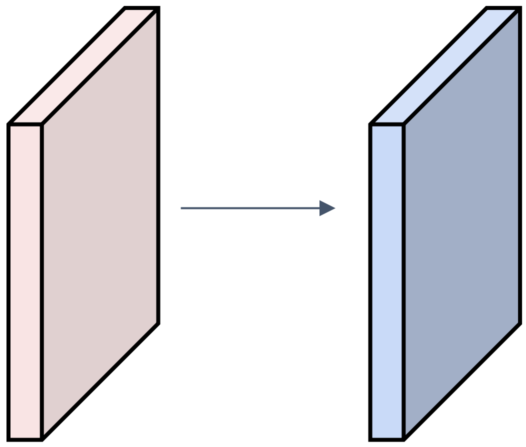
# Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

# Convolution Example



Input volume: 3 x 32 x 32

10 5x5 filters with stride 1, pad 2

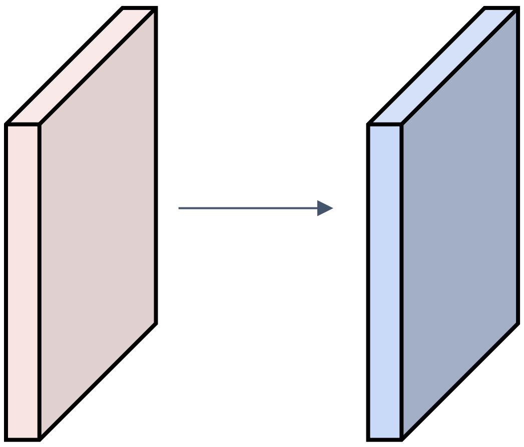
Output volume size: 10 x 32 x 32

Number of learnable parameters: **760**

Parameters per filter: **3*5*5** + 1 (for bias) = **76**

**10** filters, so total is **10** * **76** = **760**

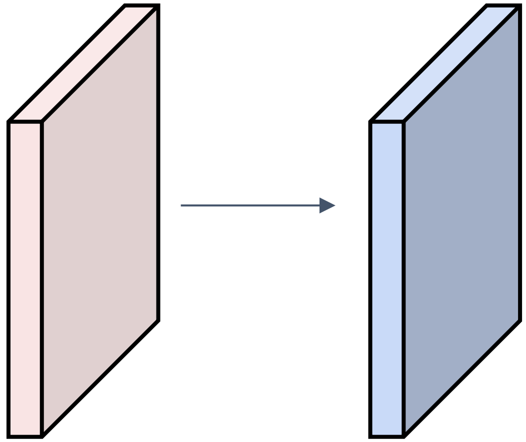
# Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?

# Convolution Example



Input volume: **3** x 32 x 32

10 **5x5** filters with stride 1, pad 2

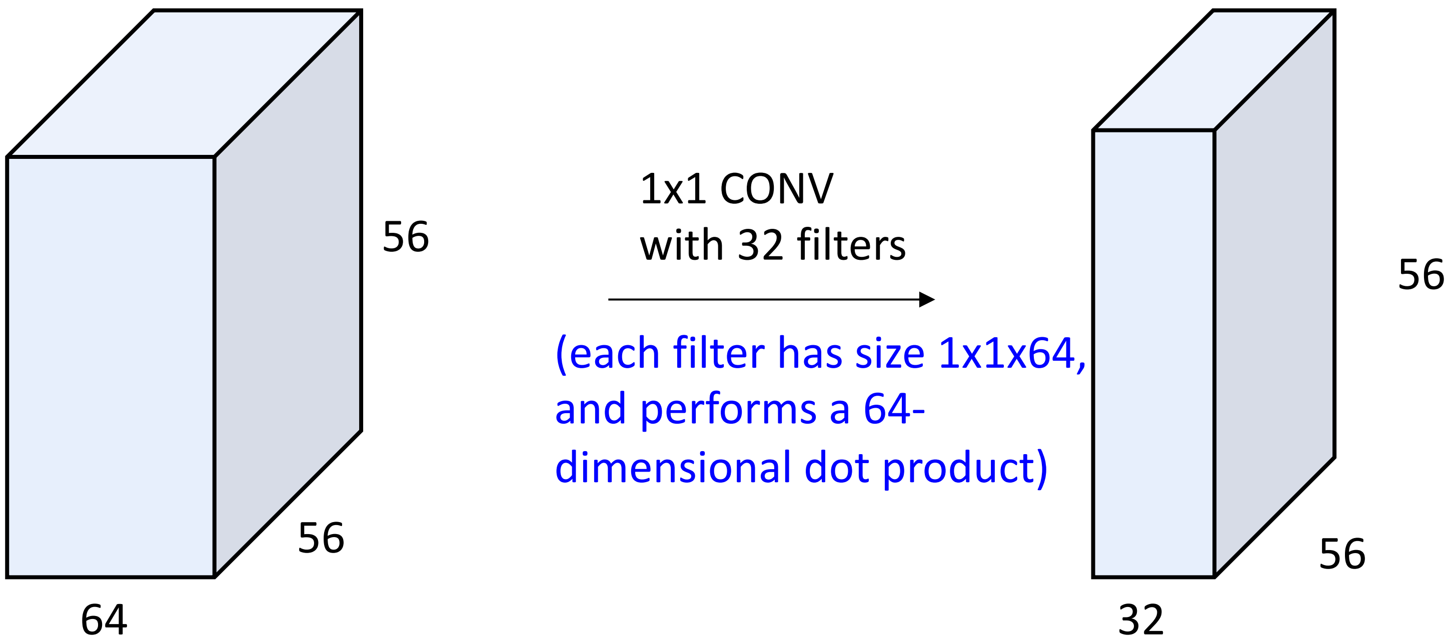
Output volume size: **10 x 32 x 32**

Number of learnable parameters: 760

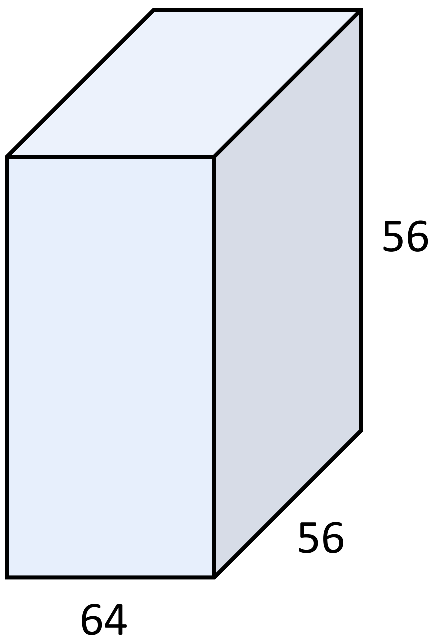
Number of multiply-add operations: **768,000**

**10*32*32** = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

# Example: 1x1 Convolution

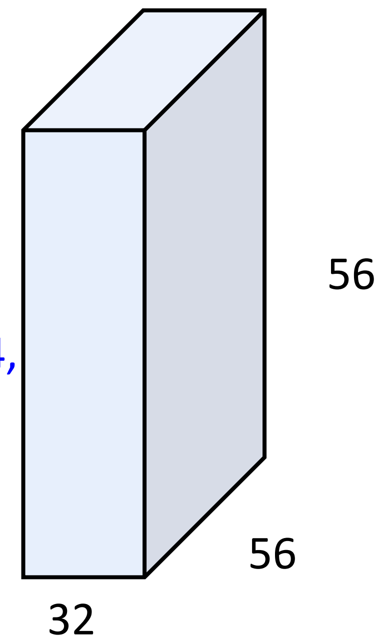


# Example: 1x1 Convolution




1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)




Stacking 1x1 conv layers gives MLP operating on each input position

Lin et al, “Network in Network”, ICLR 2014

# Convolution Summary

**Input**: $C_{in}$ x H x W
**Hyperparameters**:
- **Kernel size**: $K_H$ x $K_W$
- **Number filters**: $C_{out}$
- **Padding**: P
- **Stride**: S

**Weight matrix**: $C_{out}$ x $C_{in}$ x $K_H$ x $K_W$
giving $C_{out}$ filters of size $C_{in}$ x $K_H$ x $K_W$
**Bias vector**: $C_{out}$
**Output size**: $C_{out}$ x H' x W' where:
- H' = (H – K + 2P) / S + 1
- W' = (W – K + 2P) / S + 1

# Convolution Summary

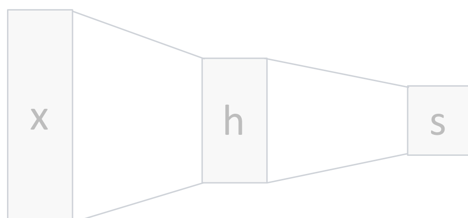
**Input**: $C_{in}$ x H x W
**Hyperparameters**:
- **Kernel size**: $K_H$ x $K_W$
- **Number filters**: $C_{out}$
- **Padding**: P
- **Stride**: S

**Weight matrix**: $C_{out}$ x $C_{in}$ x $K_H$ x $K_W$ giving $C_{out}$ filters of size $C_{in}$ x $K_H$ x $K_W$
**Bias vector**: $C_{out}$
**Output size**: $C_{out}$ x H' x W' where:
- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

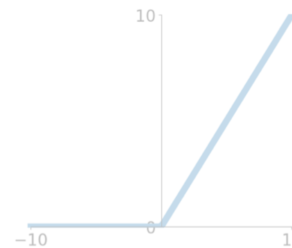
Common settings:
$K_H = K_W$ (Small square filters)
$P = (K - 1) / 2$ ("Same" padding)
$C_{in}$, $C_{out}$ = 32, 64, 128, 256 (powers of 2)
K = 3, P = 1, S = 1 (3x3 conv)
K = 5, P = 2, S = 1 (5x5 conv)
K = 1, P = 0, S = 1 (1x1 conv)
K = 3, P = 1, S = 2 (Downsample by 2)

# Components of a Convolutional Network

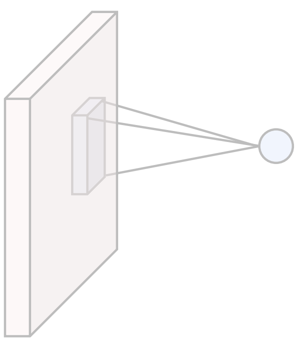
Fully-Connected Layers



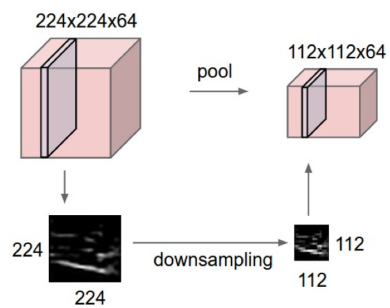

Activation Function




Convolution Layers



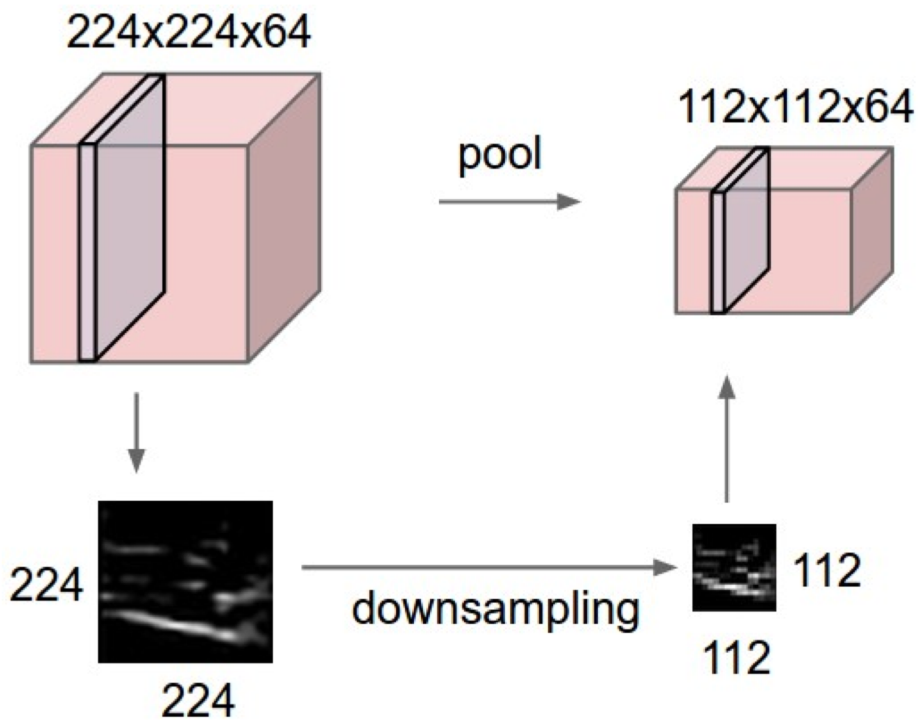
Pooling Layers




Normalization

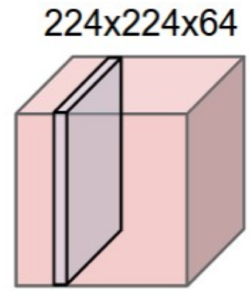
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# **Pooling Layers**: Another way to downsample

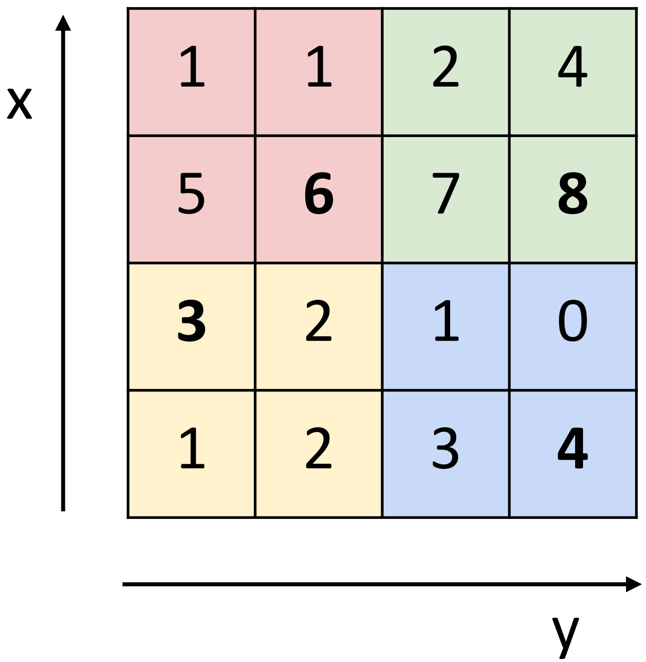



**Hyperparameters**:
Kernel Size
Stride
Pooling function

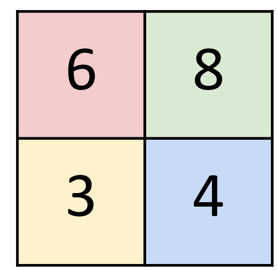
# Max Pooling




Single depth slice




Max pooling with 2x2
kernel size and stride 2




Introduces **invariance** to
small spatial shifts
No learnable parameters!

# Pooling Summary

**Input**: C x H x W
**Hyperparameters**:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

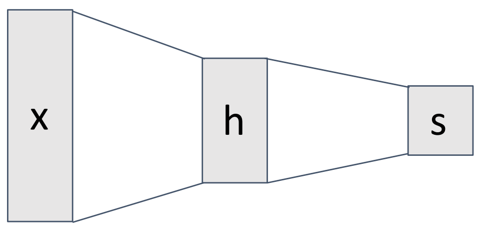
**Output**: C x H' x W' where
- $H' = (H - K) / S + 1$
- $W' = (W - K) / S + 1$

**Learnable parameters**: None!

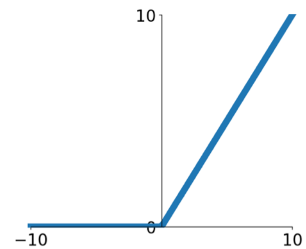
Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

# Components of a Convolutional Network

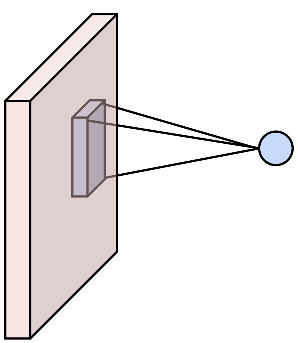
## Fully-Connected Layers



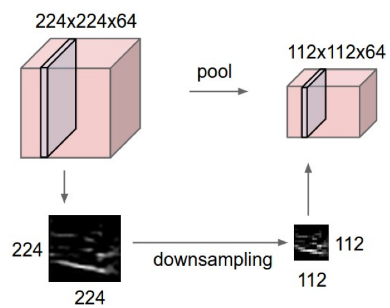

## Activation Function




## Convolution Layers



## Pooling Layers



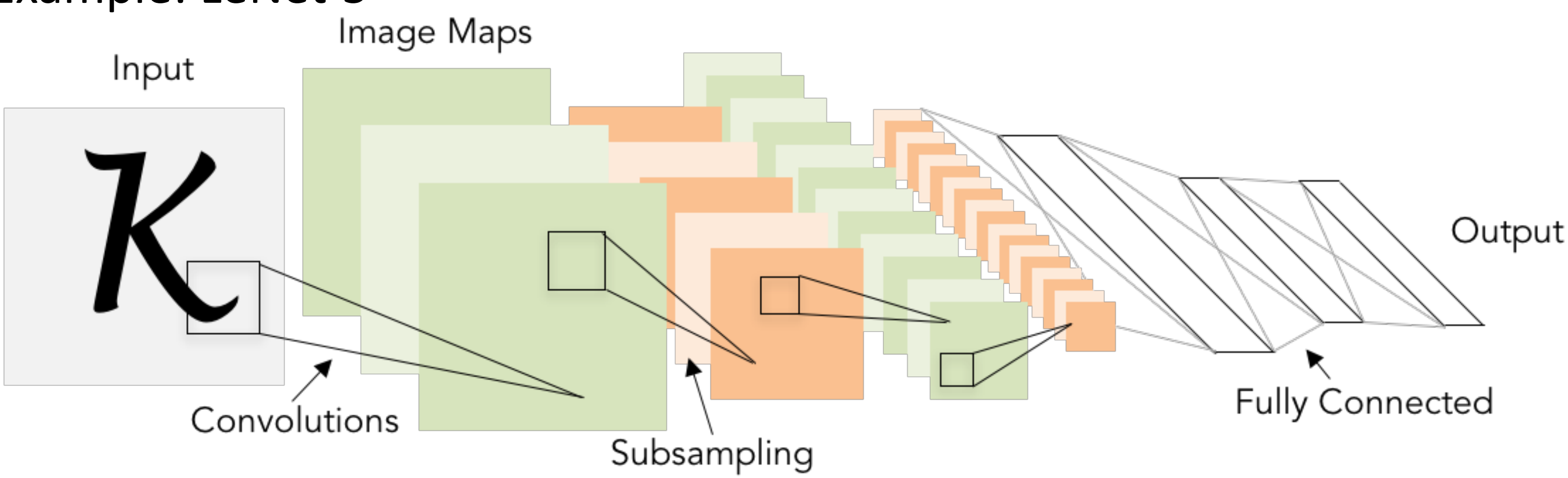

## Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Convolutional Networks

Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

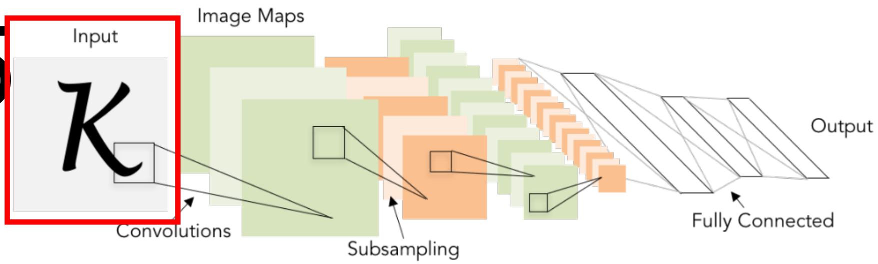
Example: LeNet-5




Lecun et al, “Gradient-based learning applied to document recognition”, 1998

# Example: LeNet-5

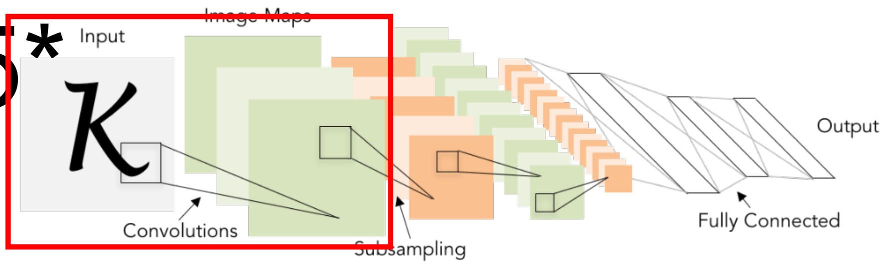
| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |




Lecun et al, “Gradient-based learning applied to document recognition”, 1998

# Example: LeNet-5*

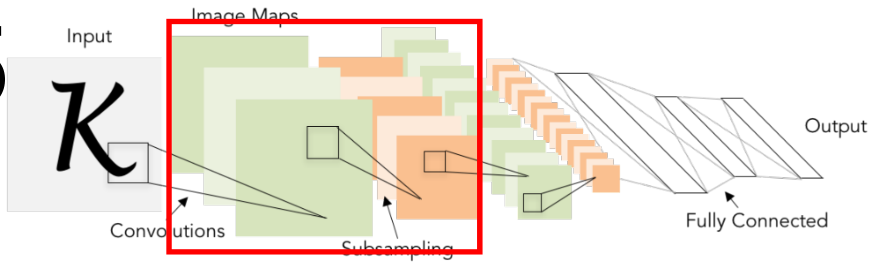
| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20*,K=5,P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU** | 20 x 28 x 28 | |




* Original paper: $C_{out}$ = 6
** Original paper: sigmoid

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5

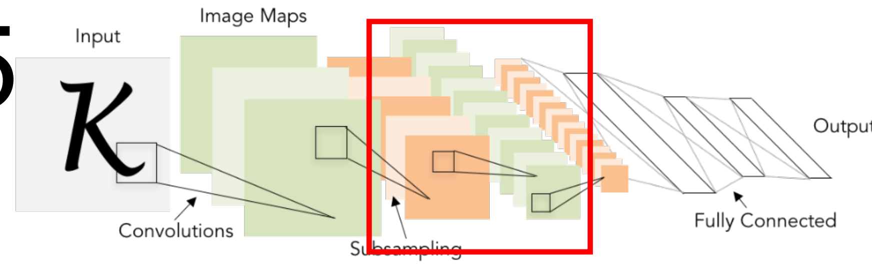



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2)* | 20 x 14 x 14 | |

* 2x2 strided convolution

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5

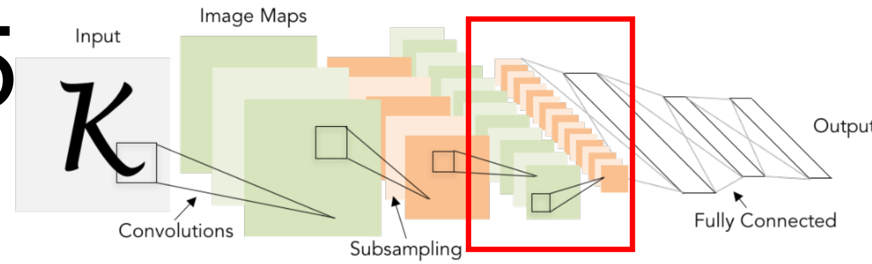



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50*,K=5,P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU** | 50 x 14 x 14 | |

* Original paper: $C_{out}$ = 16, grouped convolutions
** Original paper: sigmoid

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5

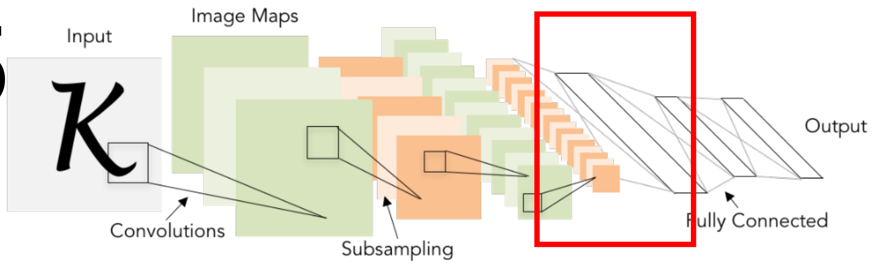



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool(K=2, S=2)* | 50 x 7 x 7 | |

* 2x2 strided convolution

Lecun et al, "Gradient-based learning applied to document recognition", 1998

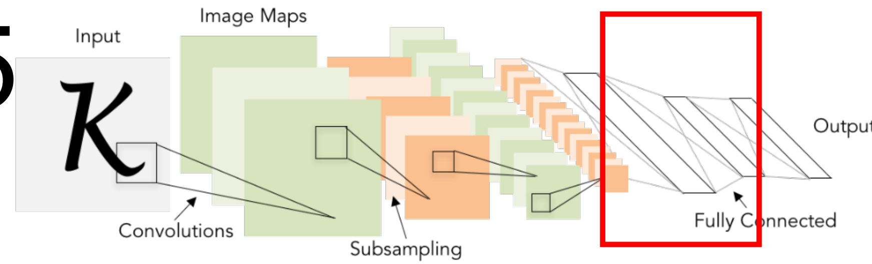
# Example: LeNet-5




| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool(K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2450 | |

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5

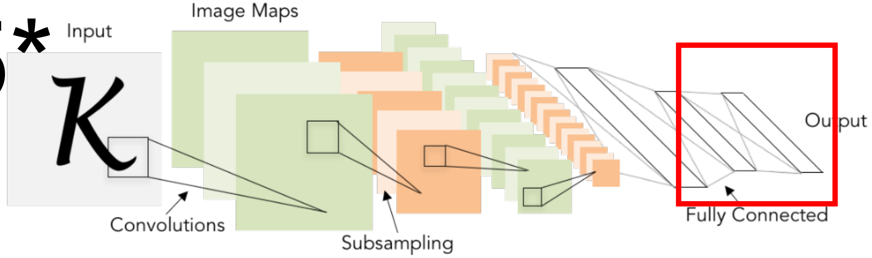



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool(K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2450 | |
| Linear (2450 -> 500) | 500 | 2450 x 500 |
| ReLU* | 500 | |

* Original paper has different 1x1 convolutions, sigmoid non-linearities

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5*

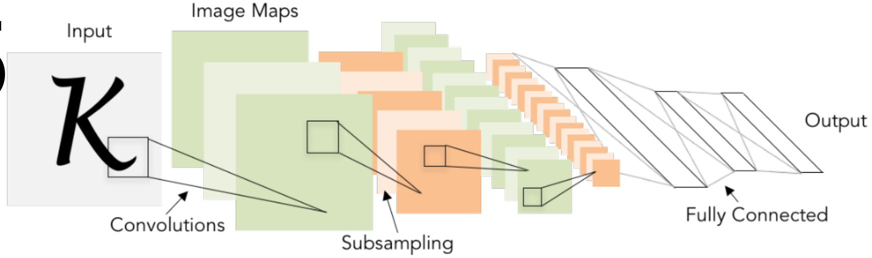



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool(K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2450 | |
| Linear (2450 -> 500) | 500 | 2450 x 500 |
| ReLU | 500 | |
| Linear (500 -> 10)* | 10 | 500 x 10 |

* Original paper uses RBF (radial basis function) kernels instead of a softmax

Lecun et al, "Gradient-based learning applied to document recognition", 1998

# Example: LeNet-5




| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool(K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool(K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2450 | |
| Linear (2450 -> 500) | 500 | 2450 x 500 |
| ReLU | 500 | |
| Linear (500 -> 10) | 10 | 500 x 10 |

As we go through the network:

Spatial size **decreases**
(using pooling or strided conv)

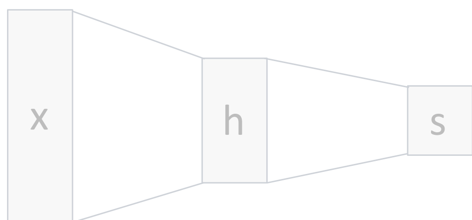
Number of channels **increases**
(total "volume" is preserved!)

Lecun et al, "Gradient-based learning applied to document recognition", 1998

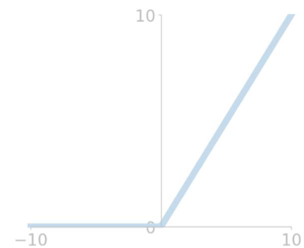
Problem: Deep Networks very hard to train!

# Components of a Convolutional Network

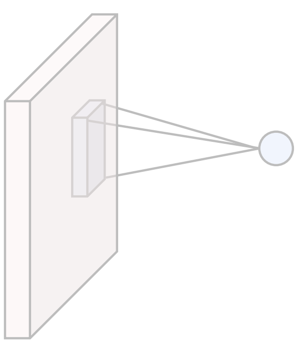
Fully-Connected Layers



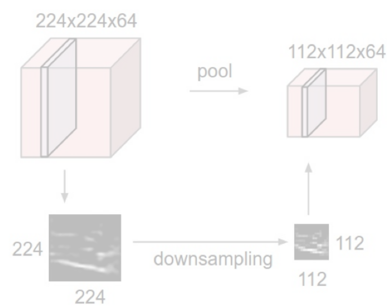

Activation Function




Convolution Layers



Pooling Layers

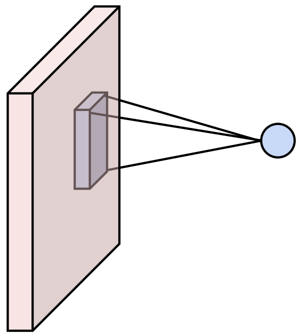



Normalization

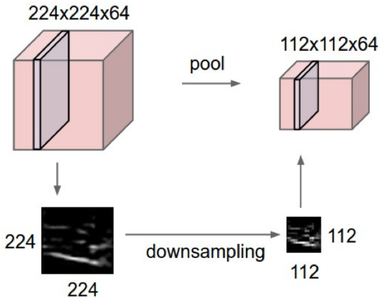
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Components of a Convolutional Network

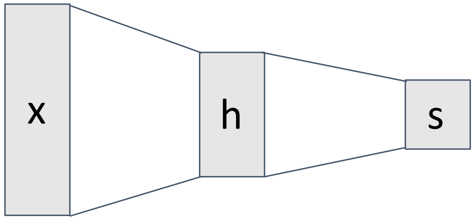
Convolution Layers



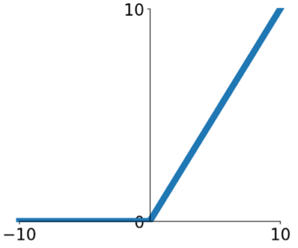
Pooling Layers




Fully-Connected Layers




Activation Function

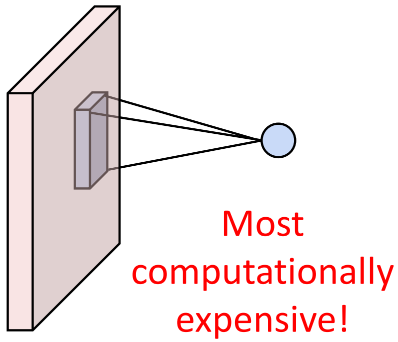



Normalization

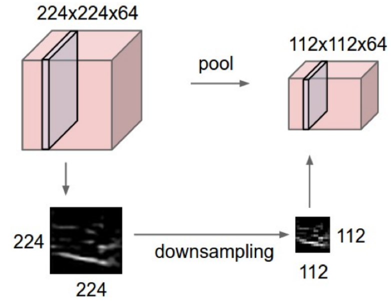
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Components of a Convolutional Network

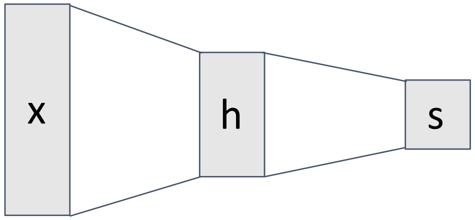
Convolution Layers



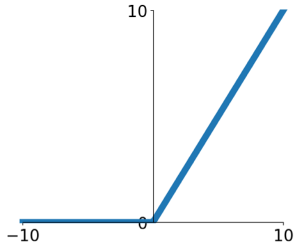

Pooling Layers




Fully-Connected Layers




Activation Function

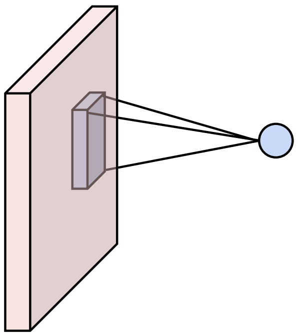



Normalization

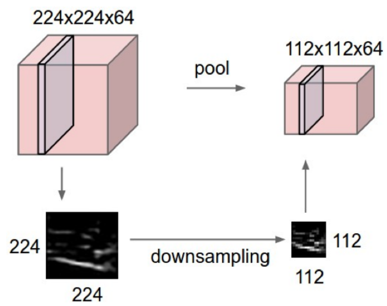
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Summary: Components of a Convolutional Network

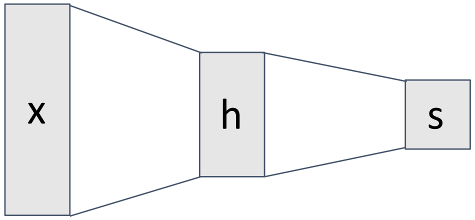
Convolution Layers



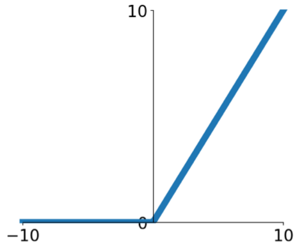
Pooling Layers




Fully-Connected Layers




Activation Function

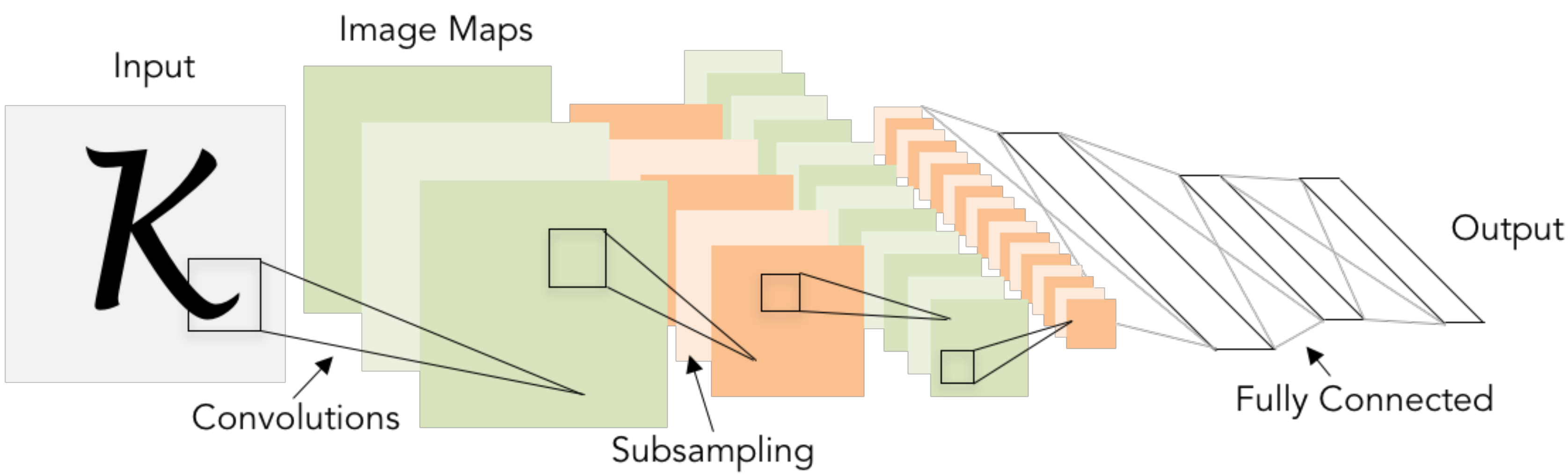



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

# Summary: Components of a Convolutional Network

**Problem**: What is the right way to combine all these components?



# Convolutional neural networks++

- Training and optimization
- More regularization (dropout,
- Convolutional neural networks
- Pooling
- Batch normalization
- CNN architectures