

Learning in Convolutional Neural Nets

Cusuh Ham and Frank Dellaert
CS 4476 Computer Vision

Many slides from Stanford's CS231N by Fei-Fei Li, Justin Johnson, Serena Yeung, as well as some slides on filtering from Devi Parikh and Kristen Grauman, who may in turn have borrowed some from others

Image Classification

Supervised Learning

CNN Review

Loss Functions

Stochastic Gradient Descent

Computing Gradients

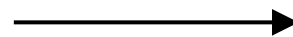
Training CNNs

Image Classification: A core task in Computer Vision



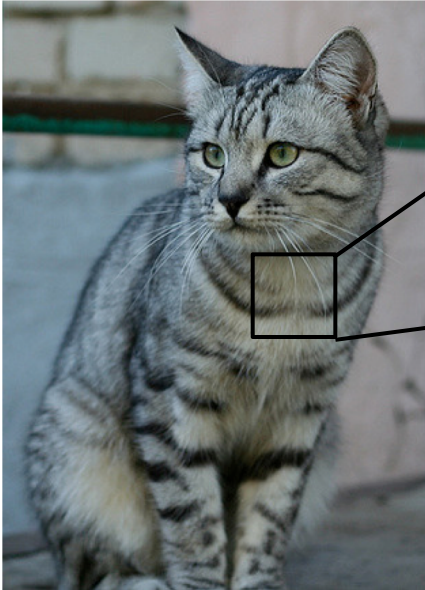
[This image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)

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



cat

The Problem: Semantic Gap



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

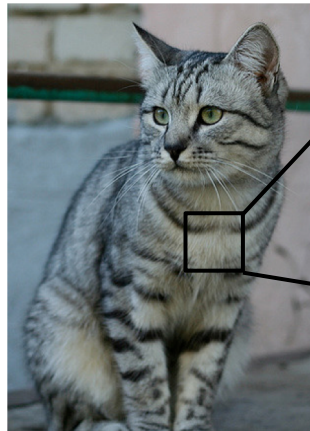
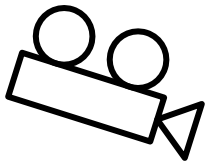
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]  
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]  
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]  
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]  
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]  
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]  
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]  
[ 89 93 90 97 106 147 131 118 113 114 113 109 106 95 77 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



[[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	185	94	85]
[76	85	90	105	128	105	87	96	95	99	115	112	106	183	99	85]
[99	81	81	93	120	131	127	100	95	98	102	99	96	93	101	94]
[106	91	61	64	69	91	88	85	101	107	109	98	75	84	96	95]	
[114	100	85	55	55	69	64	54	64	87	112	123	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	140	137	119	121	117	94	65	79	80	65	54	64	72	90]	
[127	125	131	147	133	127	126	131	111	96	89	75	61	64	72	84]	
[115	114	109	123	150	148	131	118	113	109	100	92	74	65	72	78]	
[89	93	98	97	100	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	69	71	62	81	120	130	135	105	81	90	110	118]
[87	65	71	87	106	95	69	65	76	130	126	107	92	94	105	112]
[118	97	82	86	117	123	116	66	41	51	95	93	89	95	102	107]	
[164	146	112	80	82	120	124	104	76	48	45	66	88	101	102	109]	
[157	170	157	120	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]	
[120	112	96	117	150	144	120	115	104	107	102	93	87	81	72	79]	
[123	107	96	86	83	112	153	149	122	109	104	75	80	107	112	99]	
[122	121	102	80	82	86	94	117	145	148	153	102	58	78	92	107]	
[122	164	148	103	71	56	78	83	93	103	119	139	102	61	69	84]	

All pixels change when the camera moves!

Challenges: Illumination



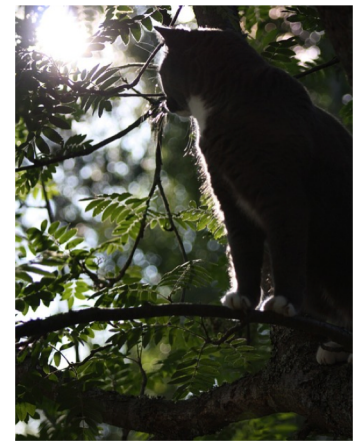
[This image](#) is [CC0 1.0](#) public domain



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



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



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

Challenges: Deformation



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



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



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



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

Challenges: Occlusion



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



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



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

Challenges: Background Clutter



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



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

Challenges: Intraclass variation



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

Image Classification

Supervised Learning

CNN Review

Loss Functions

Stochastic Gradient Descent

Computing Gradients

Training CNNs

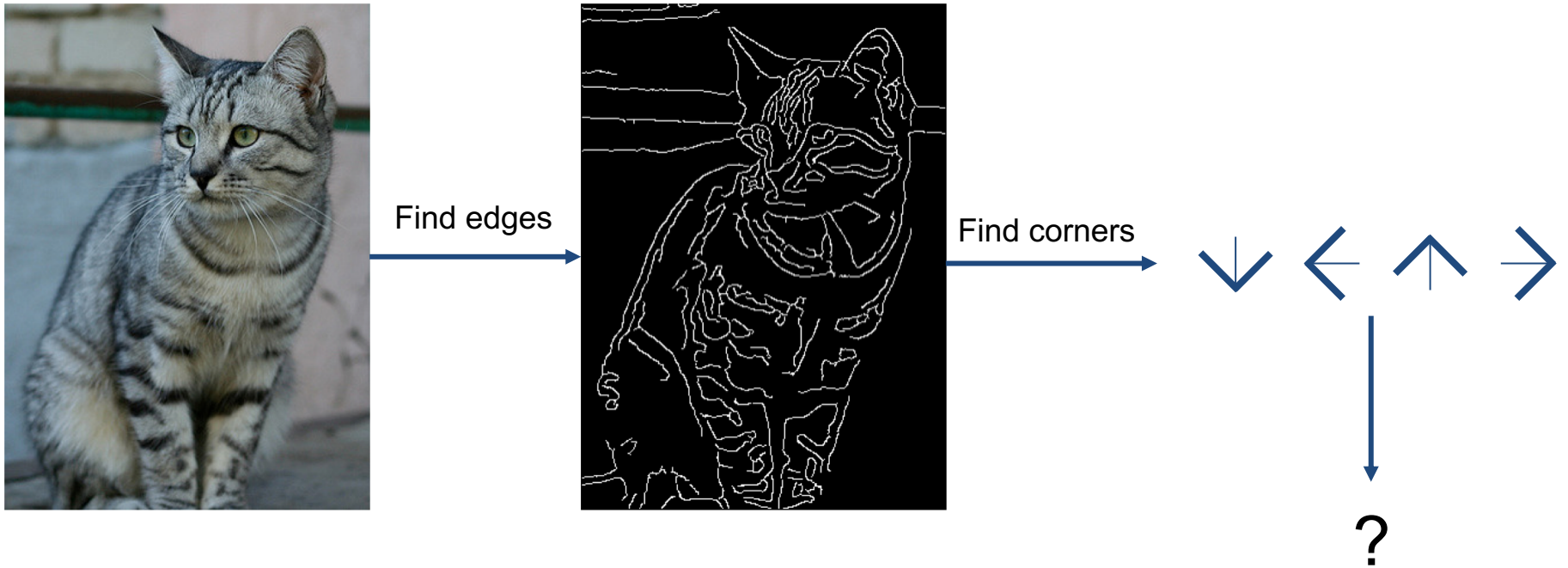
An image classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts have been made



John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

ML: A Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):  
    # Machine learning!  
    return model
```

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

Example training set

airplane



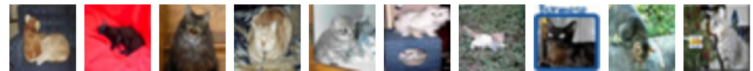
automobile



bird



cat



deer



Supervised Learning

- Input: x (images, text, emails...)
- Output: y (spam or non-spam...)
- (Unknown) Target Function
 - $f: X \rightarrow Y$ (the “true” mapping / reality)
- Data
 - $\{ (x_1, y_1), (x_2, y_2), \dots, (x_N, y_N) \}$

Procedural View

- Training Stage:
 - Training Data $\{ (x_i, y_i) \} \rightarrow h$ (Learning)
- Testing Stage
 - Test Data $x \rightarrow h(x)$ (Apply function, Evaluate error)

Image Classification
Supervised Learning

CNN Review

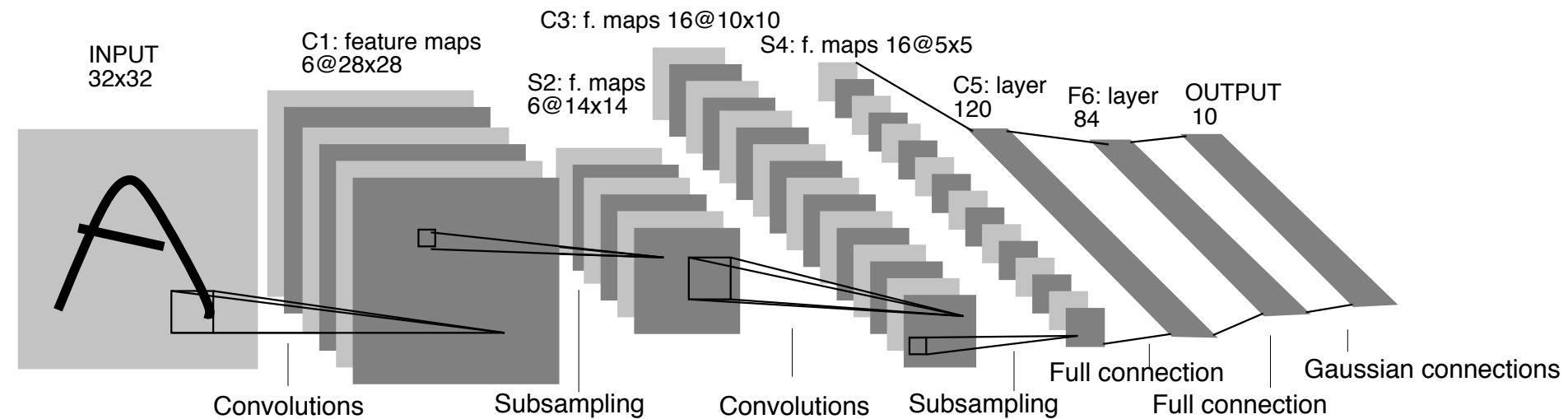
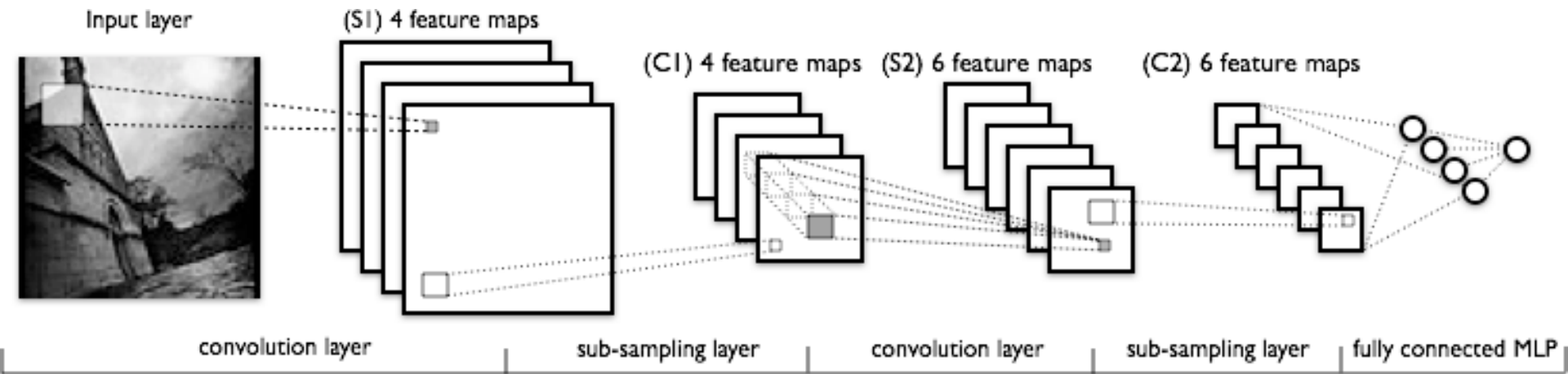
Loss Functions

Stochastic Gradient Descent

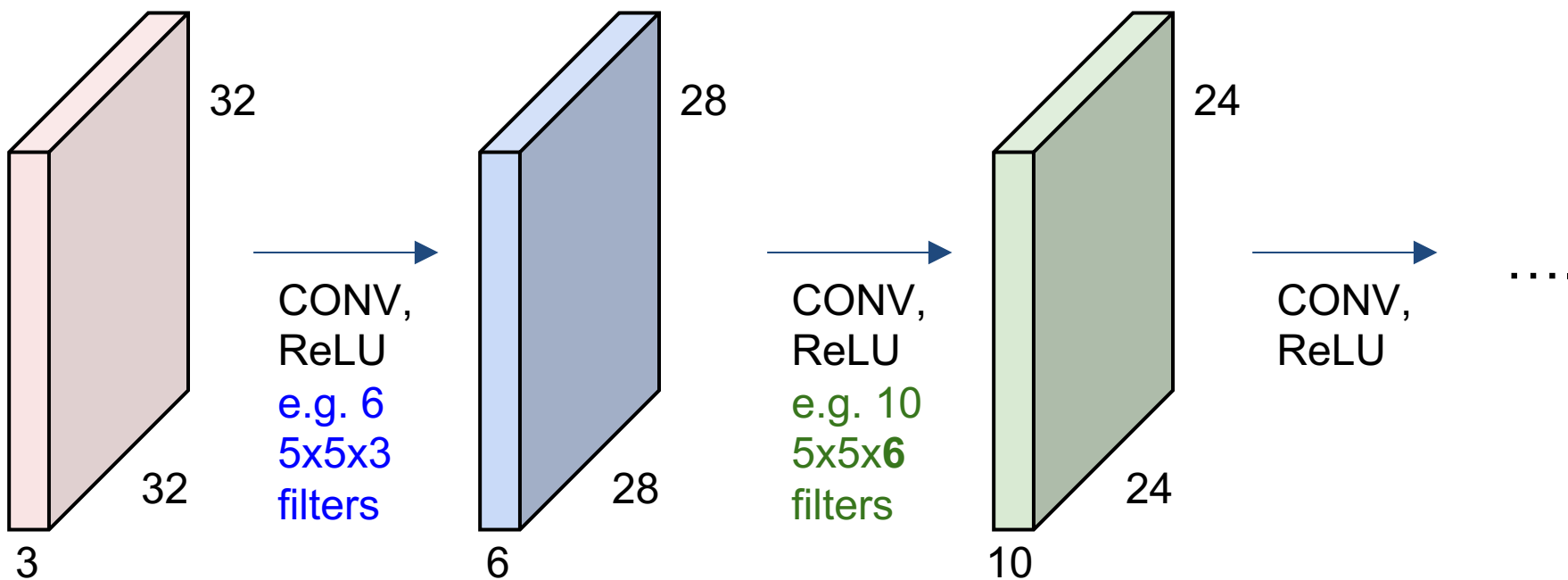
Computing Gradients

Training CNNs

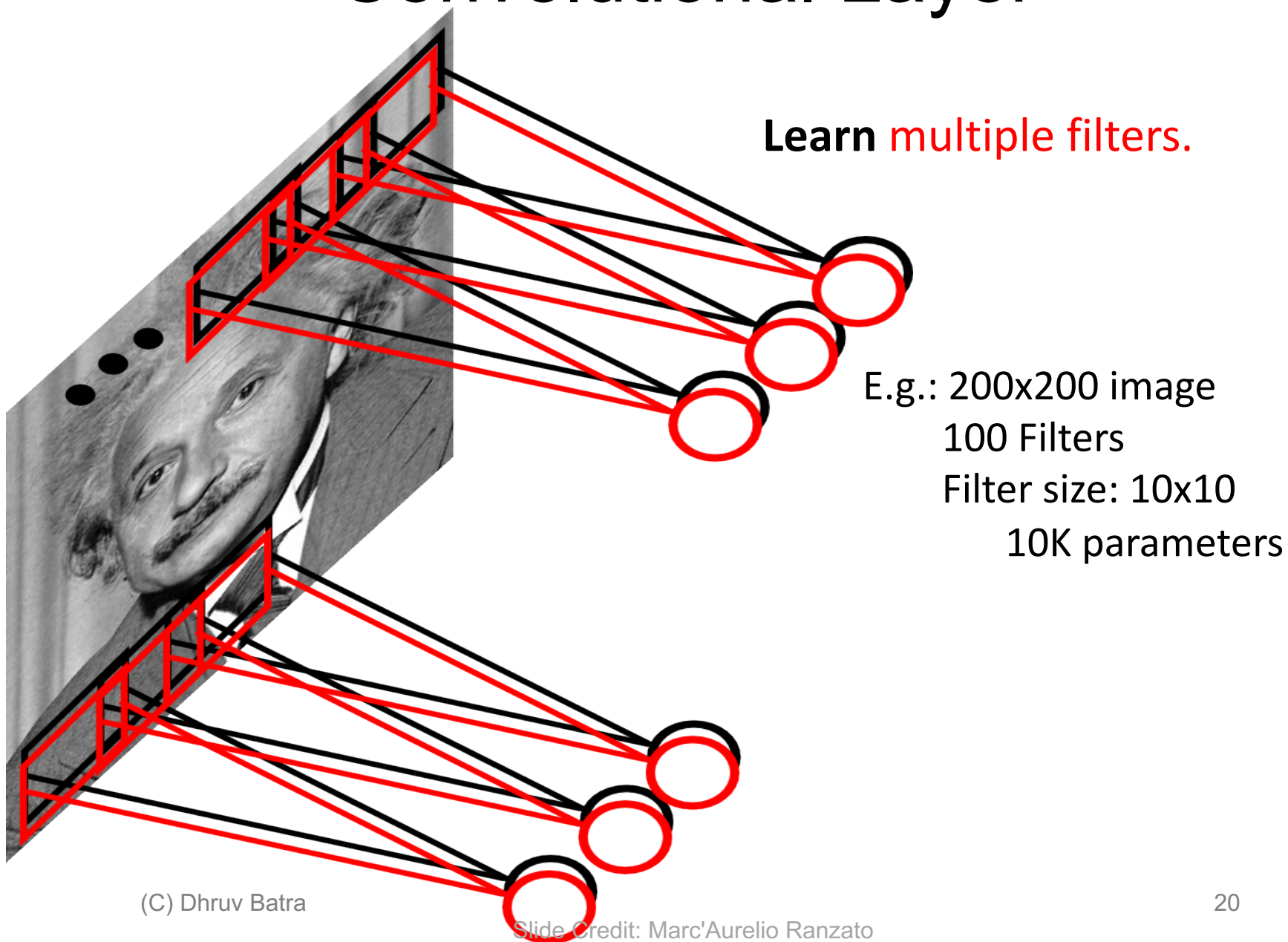
Review: CNNs



CNN or ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Convolutional Layer



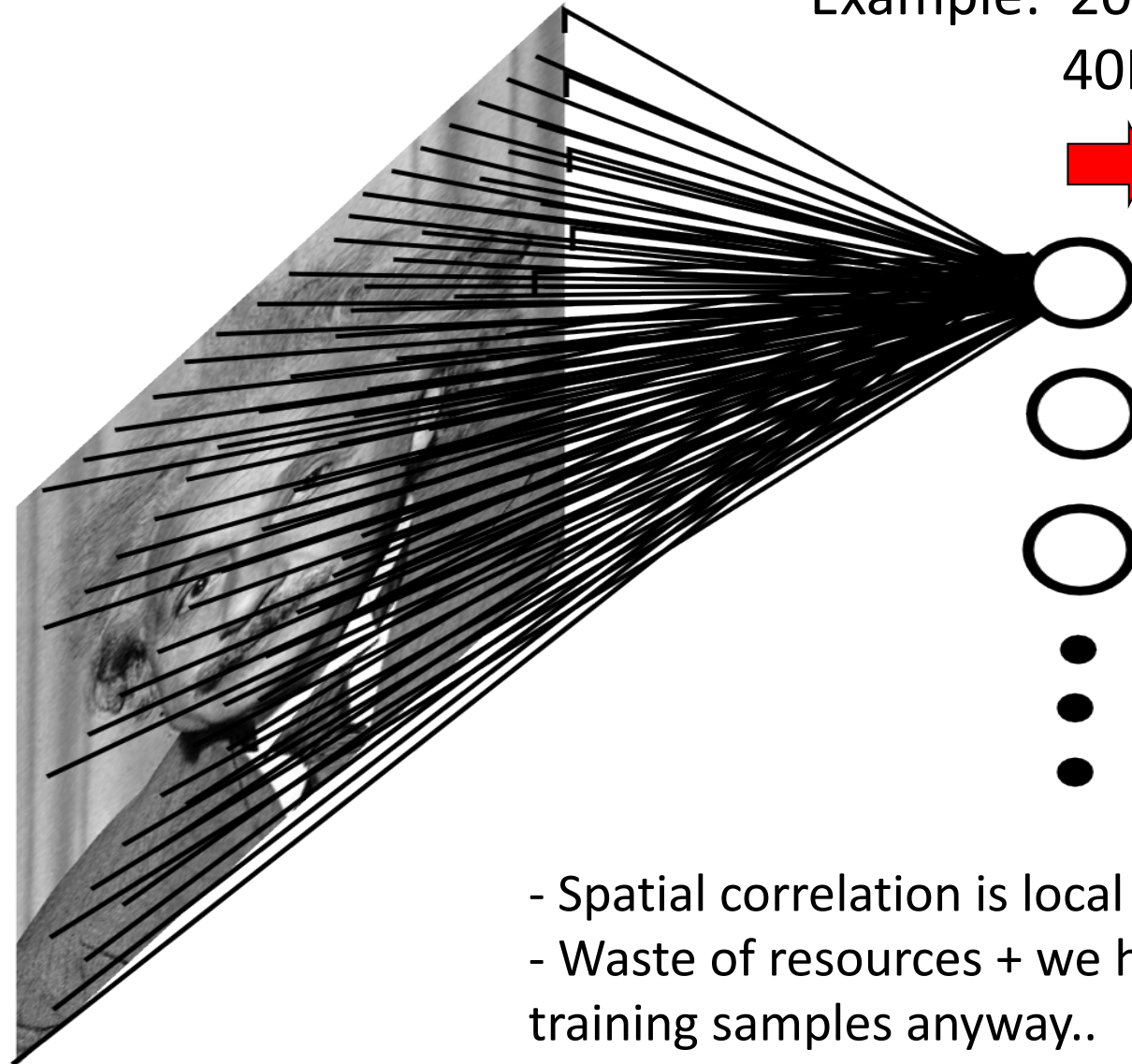
Fully Connected Layer

Example: 200x200 image

40K hidden units



~2B parameters!!!

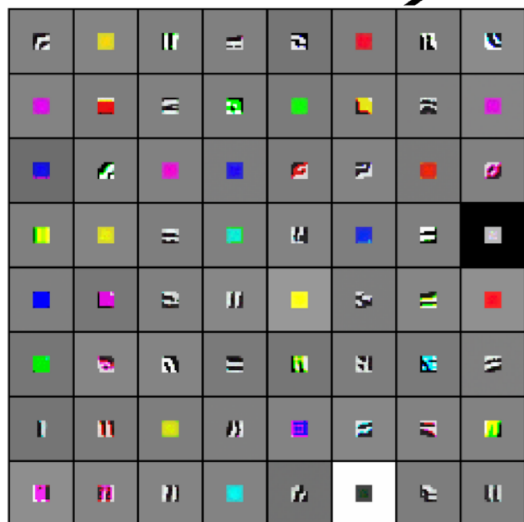
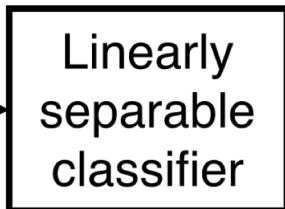
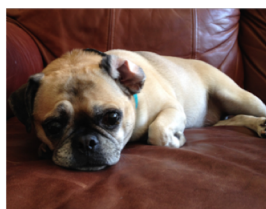


- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

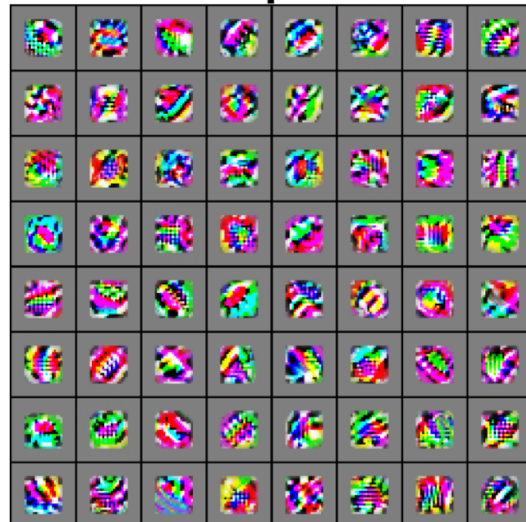
Preview

[Zeiler and Fergus 2013]

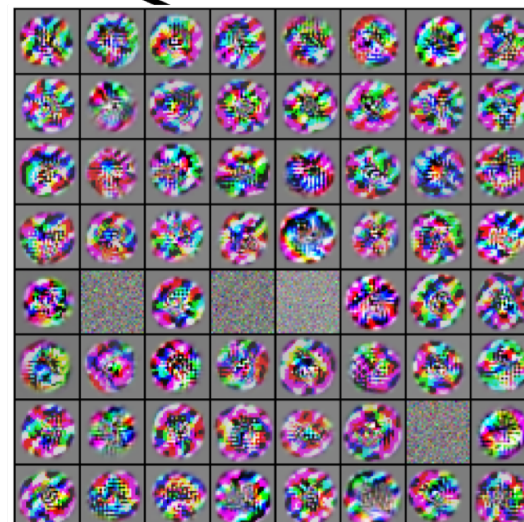
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



VGG-16 Conv1_1



VGG-16 Conv3_2



VGG-16 Conv5_3

Demo Time: a very simple NN

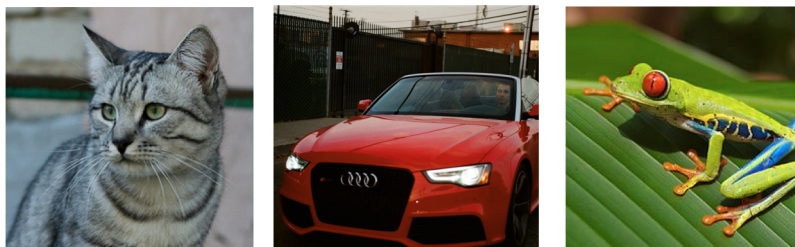
- <https://playground.tensorflow.org>

Image Classification
Supervised Learning
CNN Review

Loss Functions

Stochastic Gradient Descent
Computing Gradients
Training CNNs

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

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

Given a dataset of examples

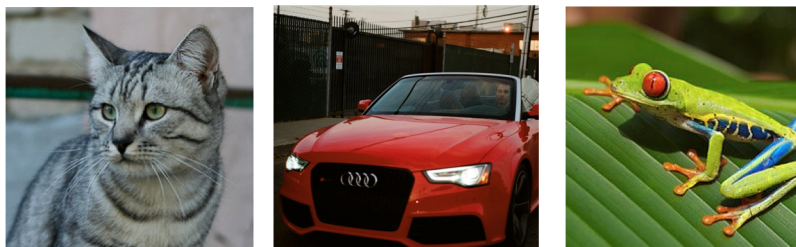
$$\{(x_i, y_i)\}_{i=1}^N$$

Where x_i is image and
 y_i is (integer) label

Loss over the dataset is a
sum of loss over examples:

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

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

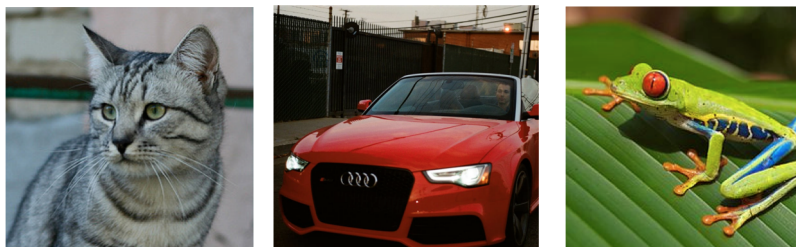
and using the shorthand for the
 scores vector: $s = f(x_i, W)$

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

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

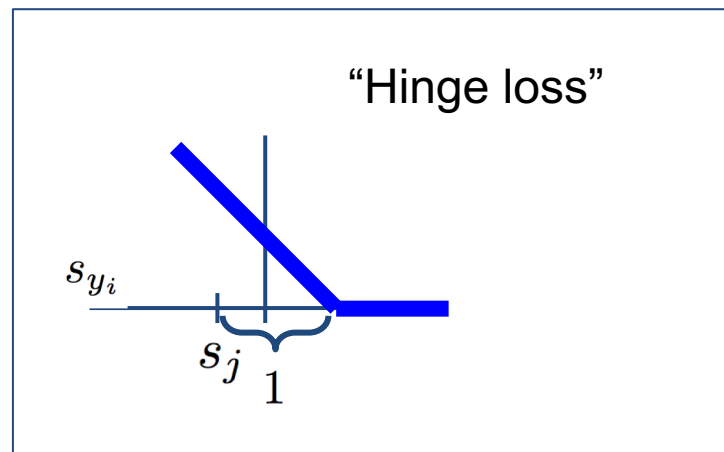
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Multiclass SVM loss:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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



Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9		

Multiclass SVM loss:

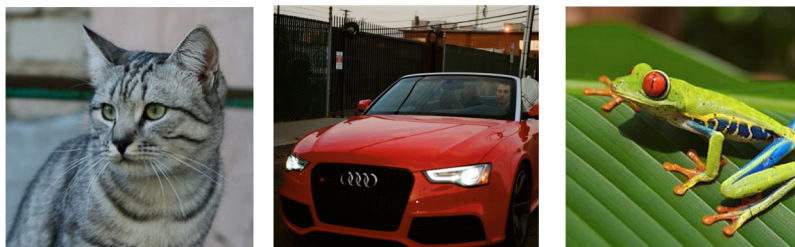
Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 5.1 - 3.2 + 1) \\
 &\quad + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 6.3) + \max(0, 6.6) \\
 &= 6.3 + 6.6 \\
 &= 12.9
 \end{aligned}$$

Suppose: 3 training examples, 3 classes.
 With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector: $s = f(x_i, W)$

the SVM loss has the form:

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

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$L = (2.9 + 0 + 12.9)/3 = 5.27$$

Image Classification
Supervised Learning

CNN Review

Loss Functions

Stochastic Gradient Descent

Computing Gradients

Training CNNs

How to minimize the loss by changing the weights?
Strategy: **Follow the slope of the loss function**



Strategy: **Follow the slope**

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

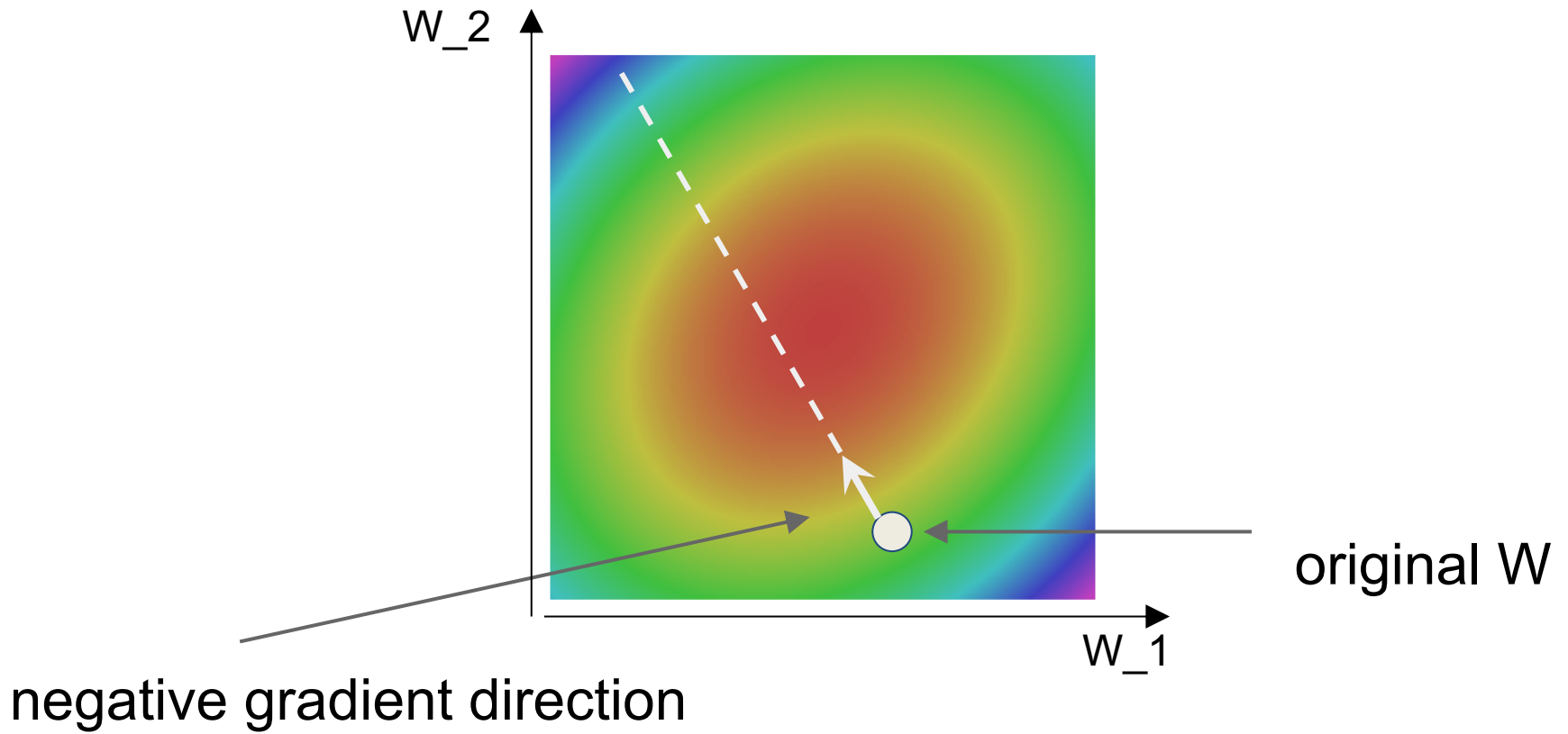
Strategy: **Follow the slope**

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**



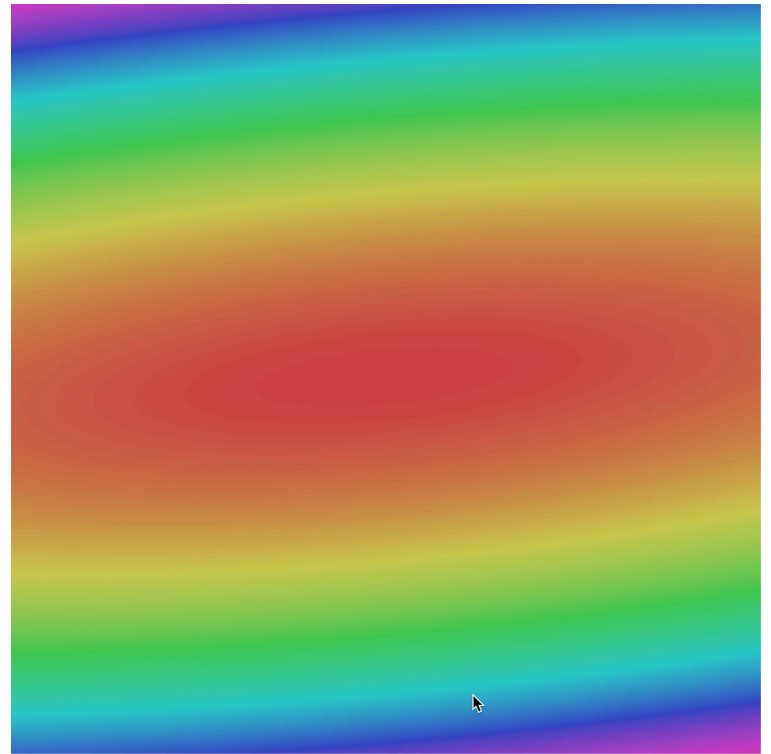
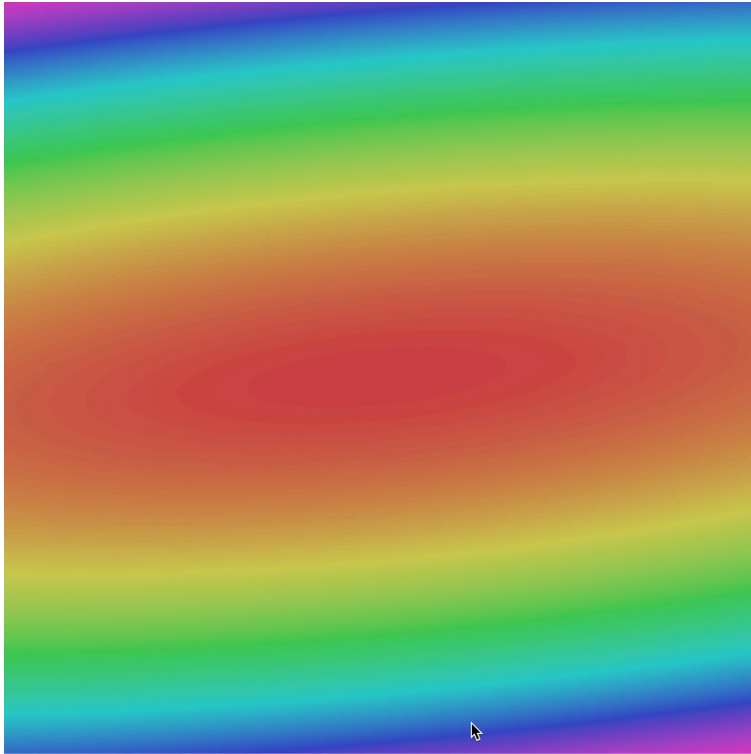
Gradient Descent

```
# Vanilla Gradient Descent
```

```
while True:
```

```
    weights_grad = evaluate_gradient(loss_fun, data, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```



Gradient Descent has a problem

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

Full sum expensive
when N is large!

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

Stochastic Gradient Descent (SGD)

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

Full sum expensive
when N is large!

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

Approximate sum
using a **minibatch** of
examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

```
    data_batch = sample_training_data(data, 256) # sample 256 examples
```

```
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```

Image Classification
Supervised Learning
CNN Review
Loss Functions
Stochastic Gradient Descent
Computing Gradients
Training CNNs

How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
 - Forward mode AD
 - Reverse mode AD
 - aka “backprop”

Numerical vs Analytic Gradients

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient.

This is called a **gradient check**.

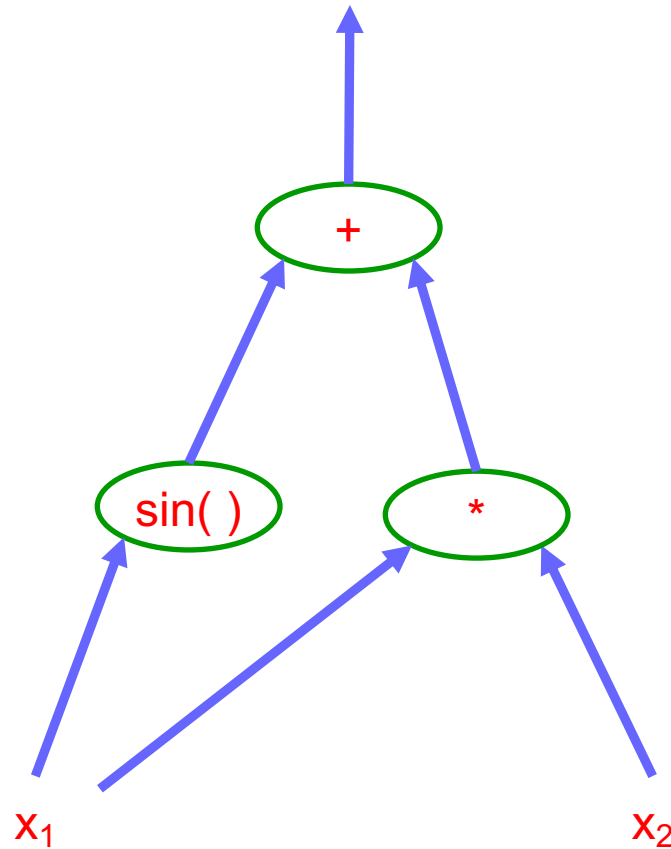
Automatic Differentiation

- Notation

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$

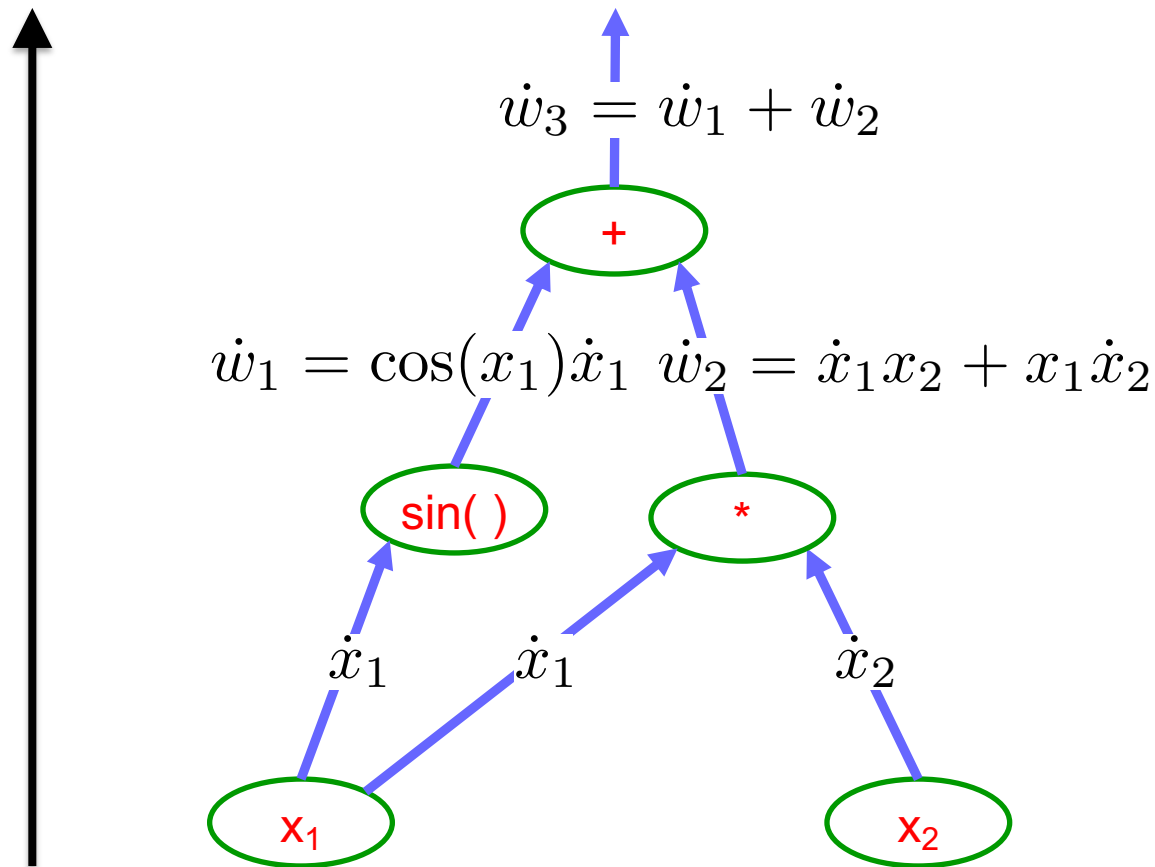
Computational Graphs

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



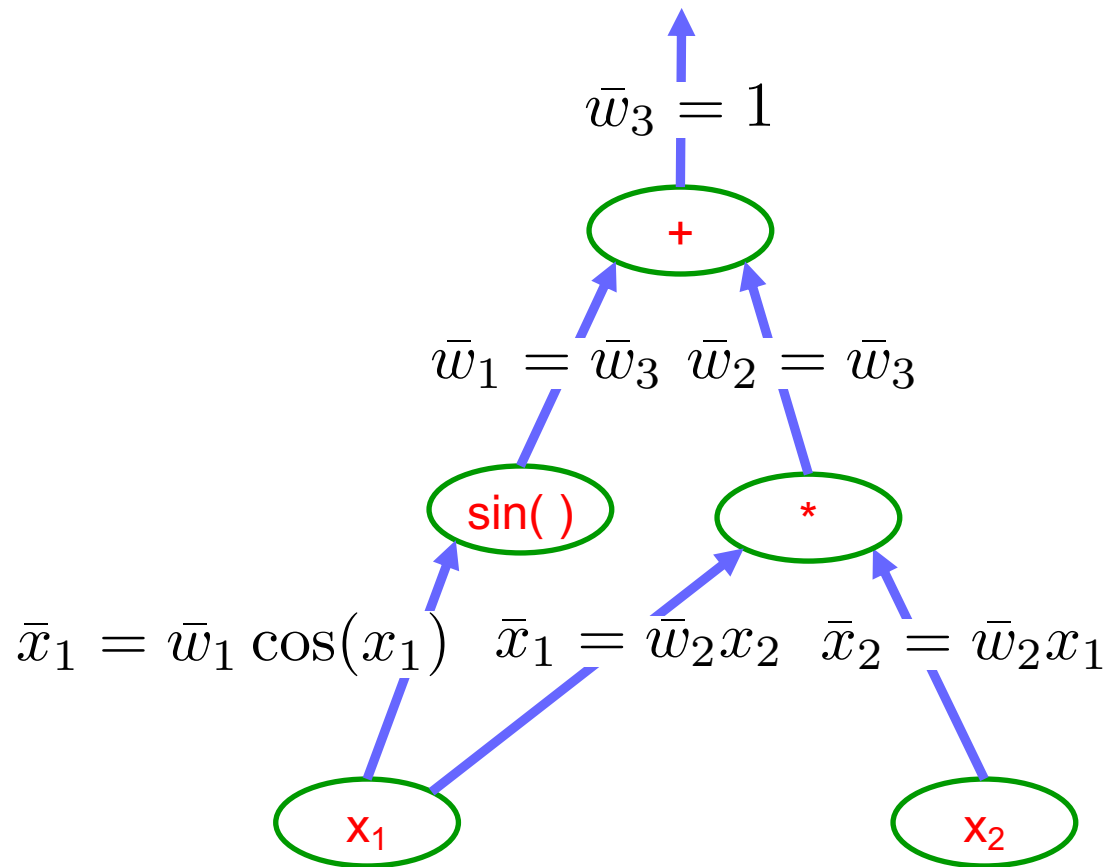
Example: Forward mode AD

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Example: Reverse mode AD

$$f(x_1, x_2) = x_1 x_2 + \sin(x_1)$$



Forward Pass vs Forward mode AD vs Reverse Mode AD

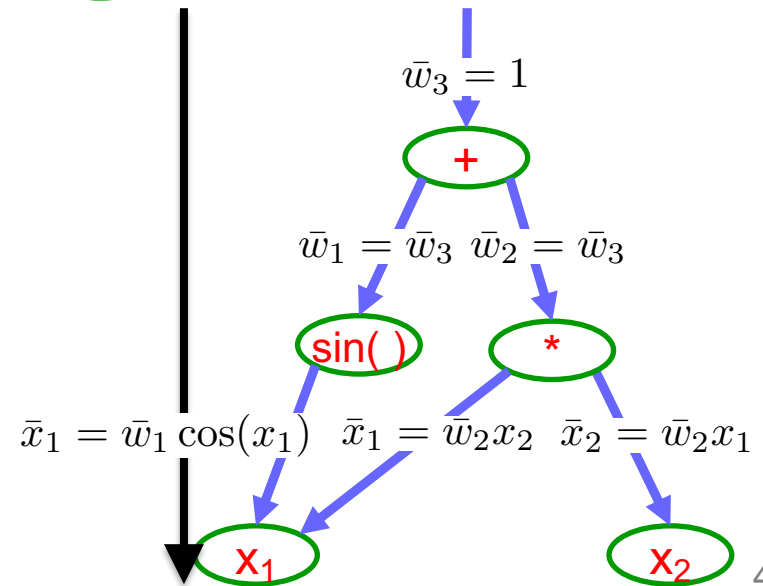
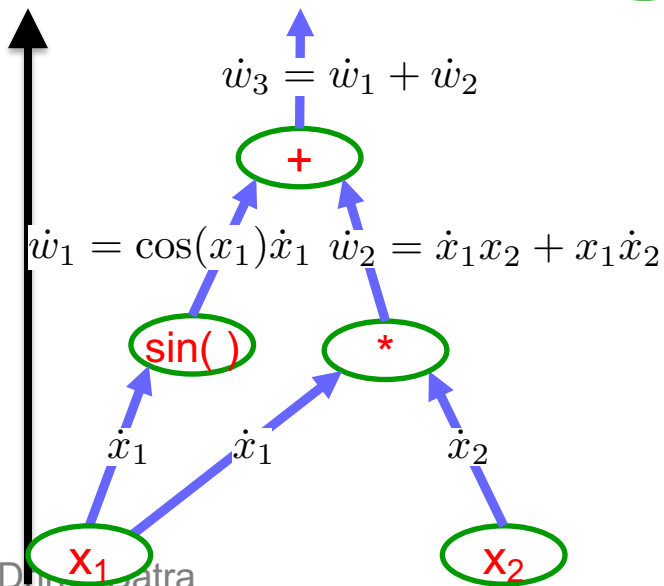
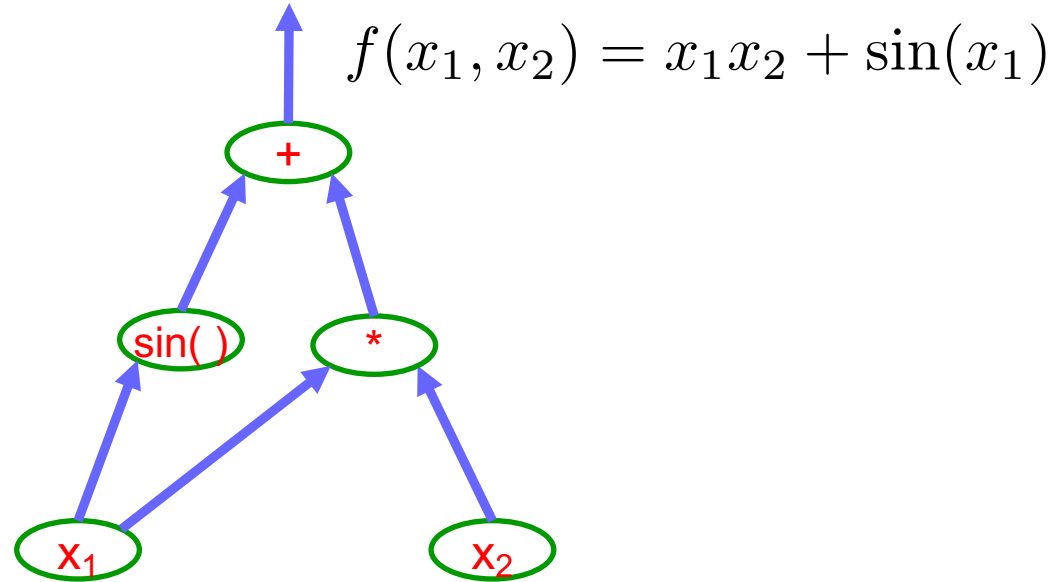
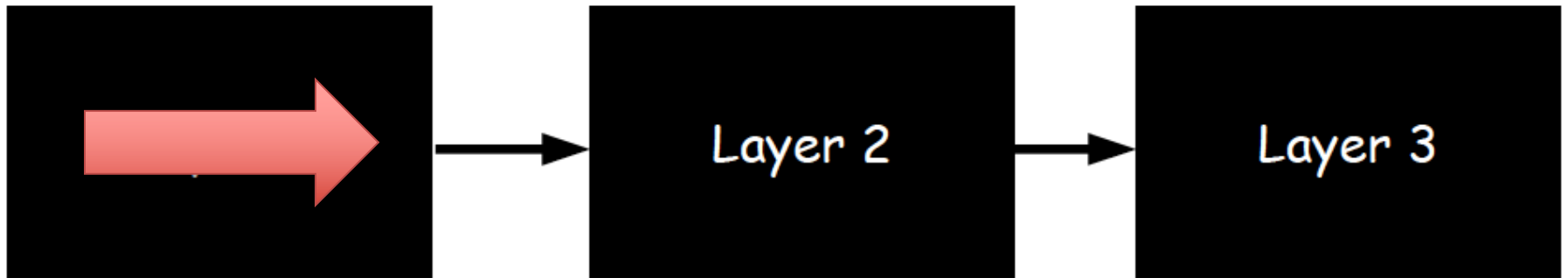


Image Classification
Supervised Learning
CNN Review
Loss Functions
Stochastic Gradient Descent
Computing Gradients
Training CNNs

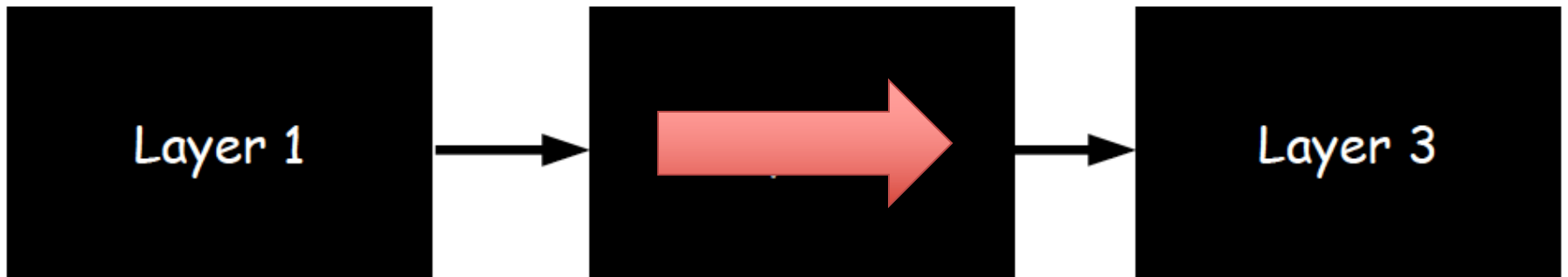
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]



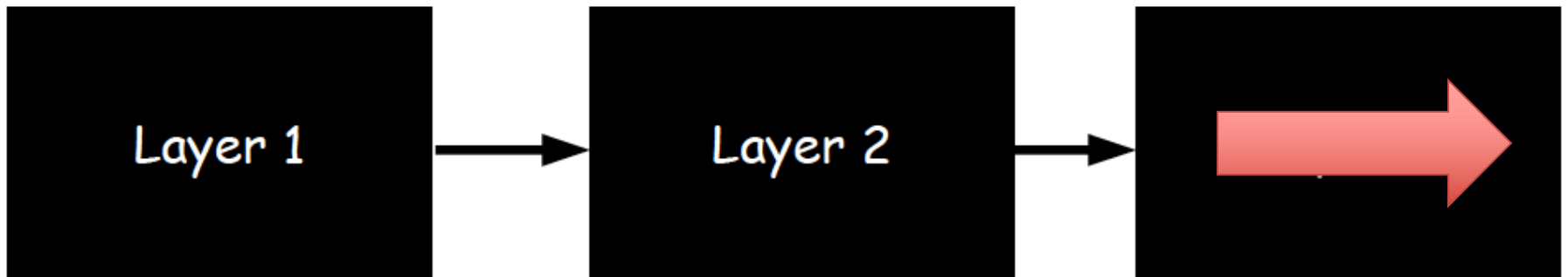
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]



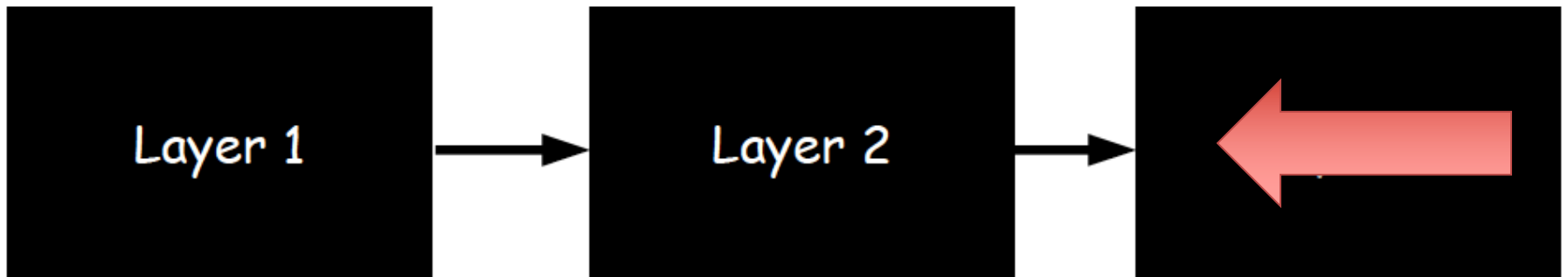
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]



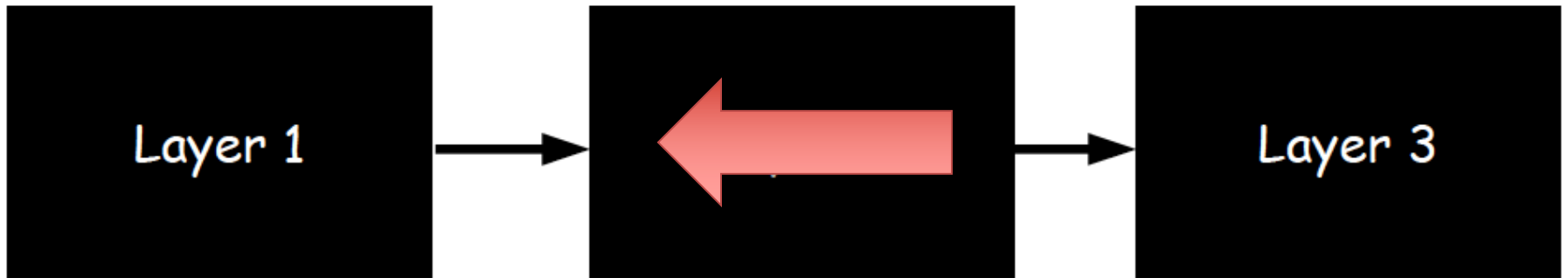
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



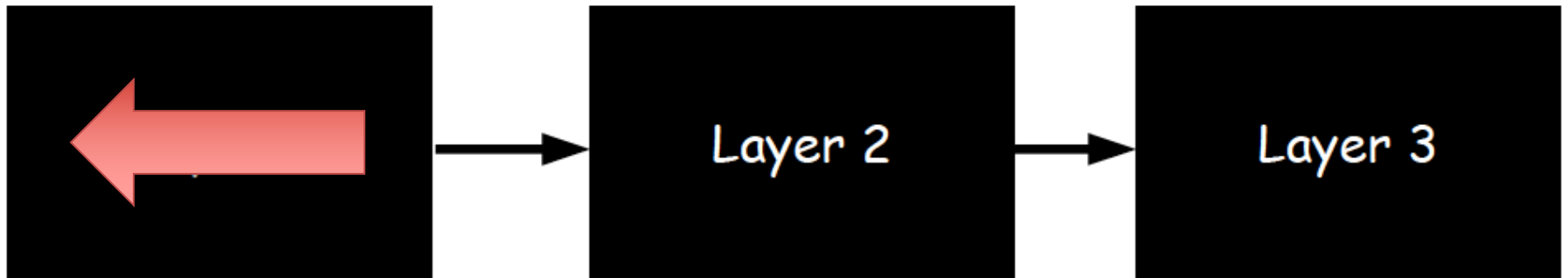
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



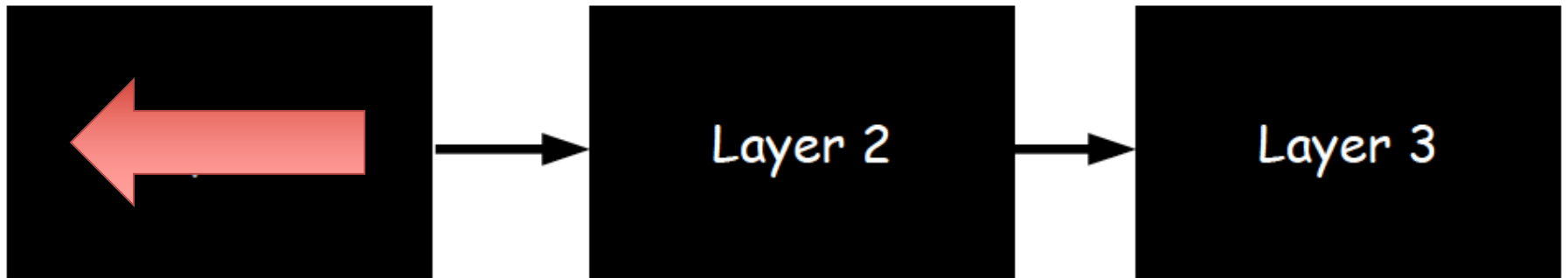
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]



Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
- Step 3: Use gradient to update parameters



$$\theta \leftarrow \theta - \eta \frac{dL}{d\theta}$$