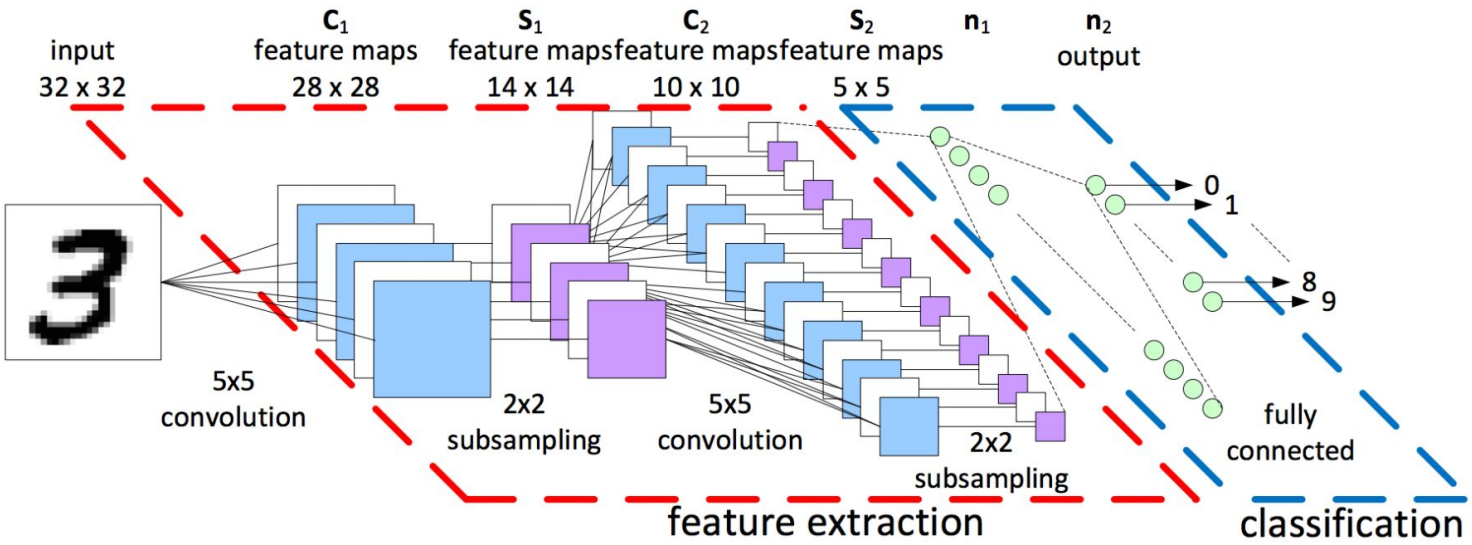
CNN Basics

Chongruo Wu

Overview



1. **Forward**: compute the output of each layer
2. **Back propagation**: compute gradient
3. **Updating**: update the parameters with computed gradient

Agenda

1. Forward

   Conv, Fully Connected, Pooing, Non-linear Function
   Loss functions

2. Back Propagation, Computing Gradient

   Chain rule

3. Updating Parameters

   SGD

4. Training

Agenda

1. Forward

   Conv, Fully Connected, Pooing, non-linear Function

   Loss functions

2. Backward, Computing Gradient

   Chain rule

3. Updating Parameters

   SGD

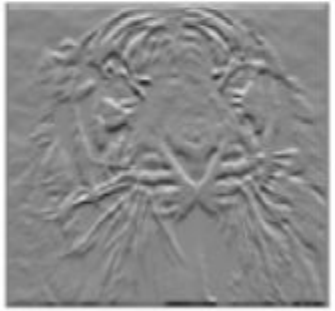
4. Training

Before Deep Learning

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

$$\frac{\partial f(x,y)}{\partial y}$$



-1	1

-1	1
1	-1

?

or

Which shows changes with respect to x?

Slide credit: Kristen Grauman

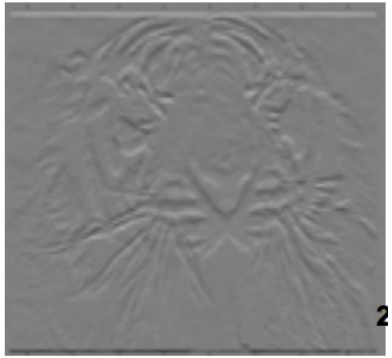
Before Deep Learning

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



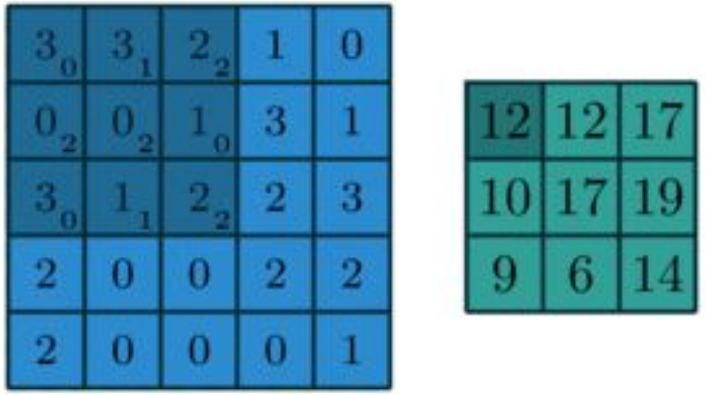
Slide credit: Kristen Grauman

Convolution (One Channel)

Convolution, One Channel

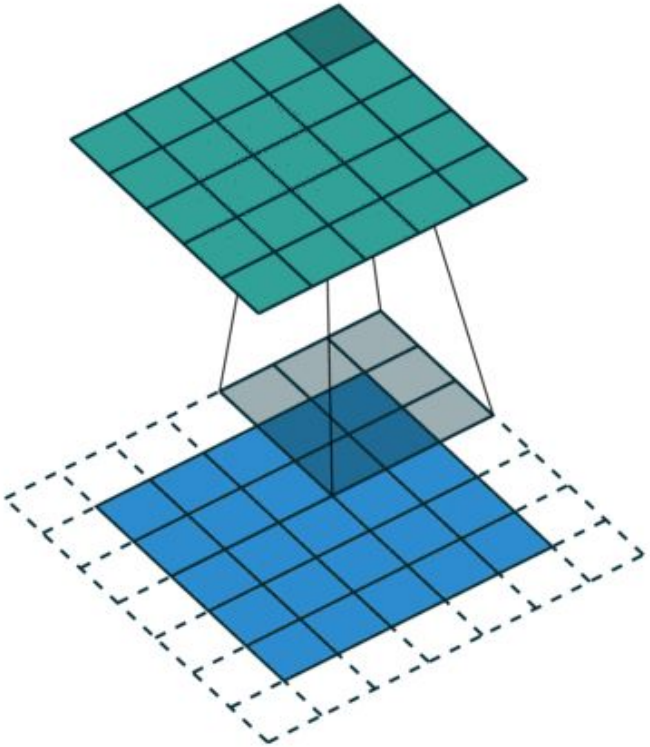
Gif Animation, https://goo.gl/2L2KdP

- Kernels are learned.
- Here we show the forward process
- After each iteration, parameters of kernels are changed.



Convolution, Padding

Gif Animation, https://goo.gl/Rm38CJ



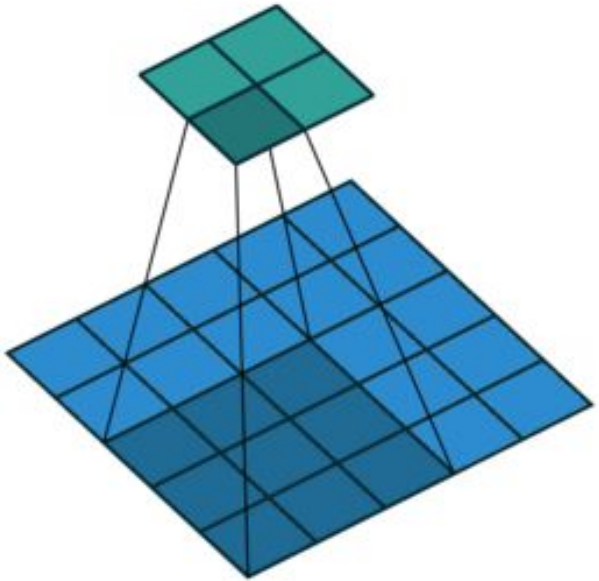
Pad = 1

output size = input size

Convolution, Stride

Gif Animation, https://goo.gl/Rm38CJ

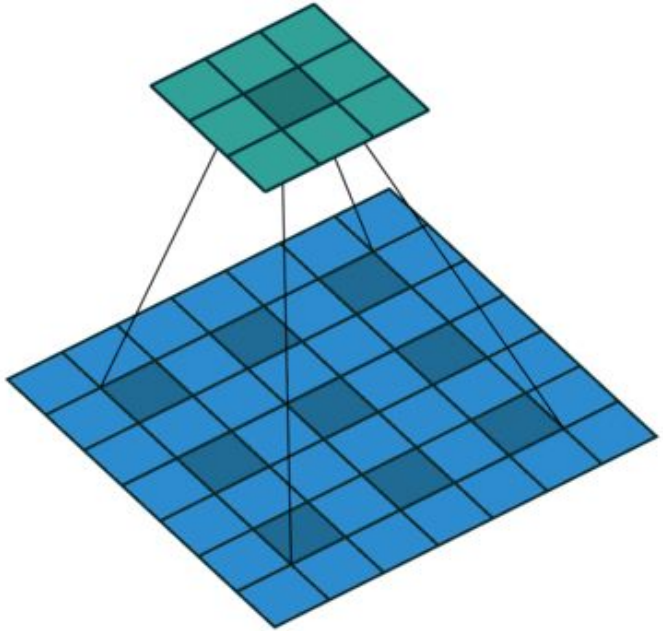
Stride > 1



Dilated Convolution

Gif Animation, https://goo.gl/Rm38CJ

Dilated Stride > 1



Convolution (Multiple Channels)

Convolution, Multiple Channels

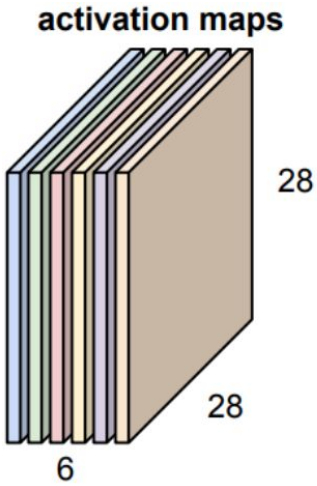
Representation for Feature Map:

(batch, channels, height, width)

For input image

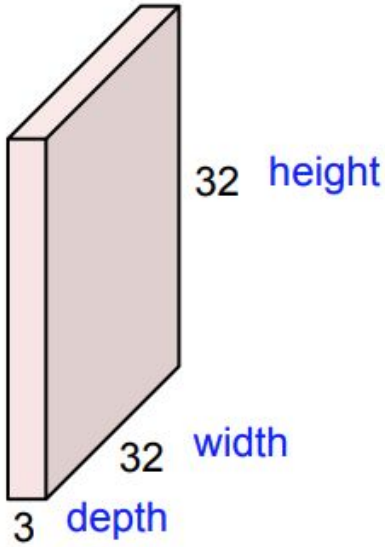
(batch, 3 , height, width)

R,G,B



Convolution Layer

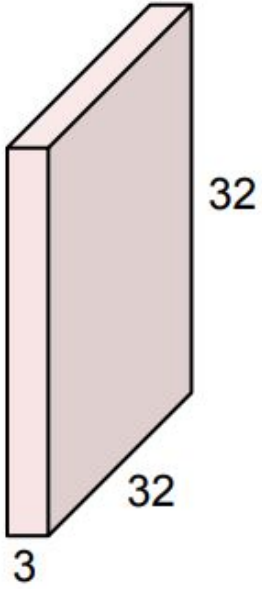
32x32x3 image



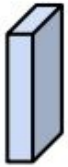
Andrej Karpathy, Bay Area Deep Learning School, 2016

Convolution Layer

32x32x3 image



5x5x3 filter



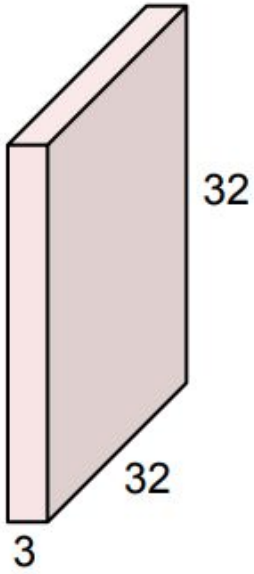
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Andrej Karpathy, Bay Area Deep Learning School, 2016

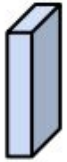
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



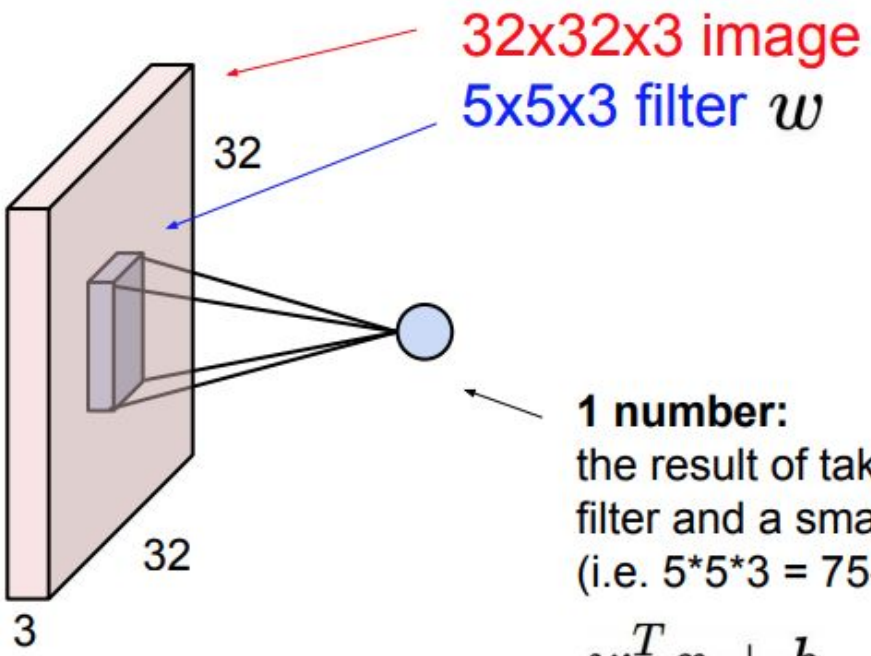
5x5x3 filter



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

Andrej Karpathy, Bay Area Deep Learning School, 2016

Convolution Layer

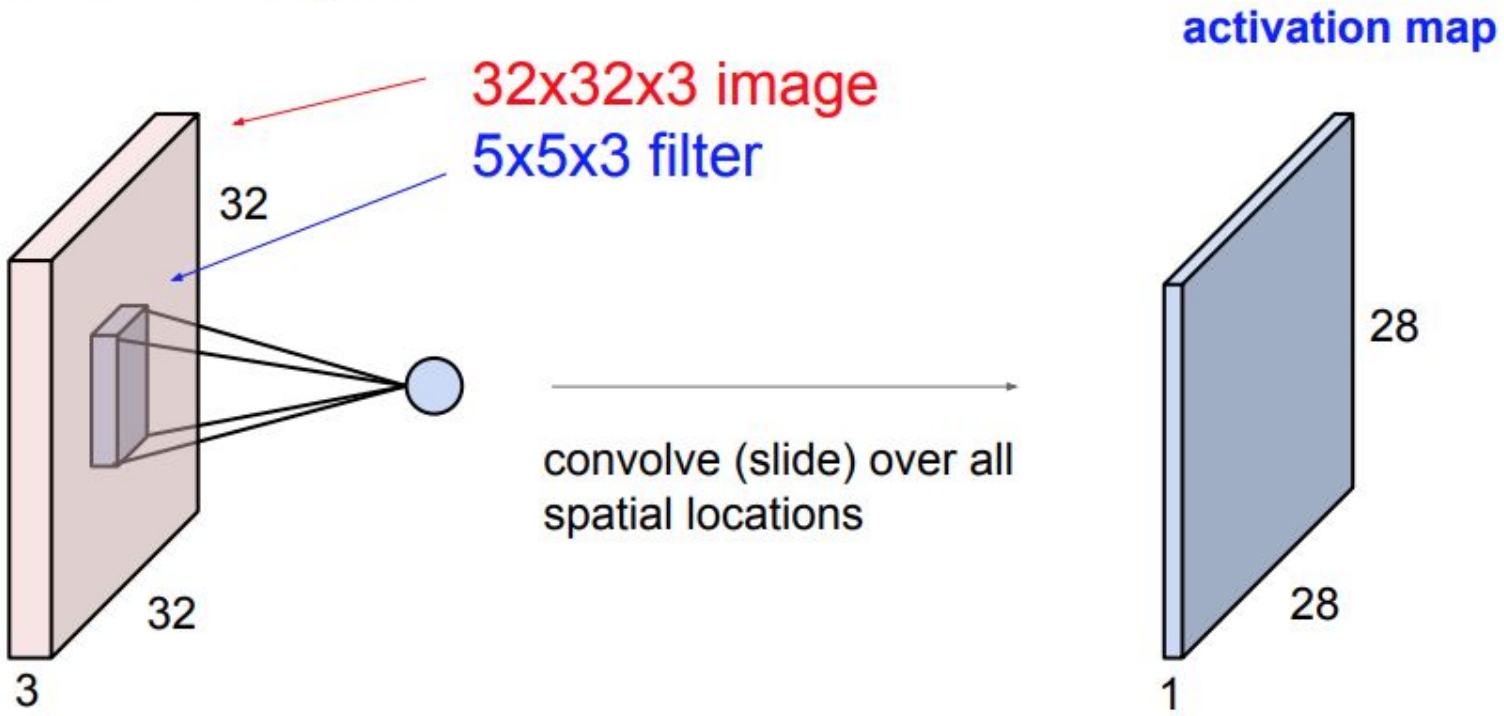


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Andrej Karpathy, Bay Area Deep Learning School, 2016

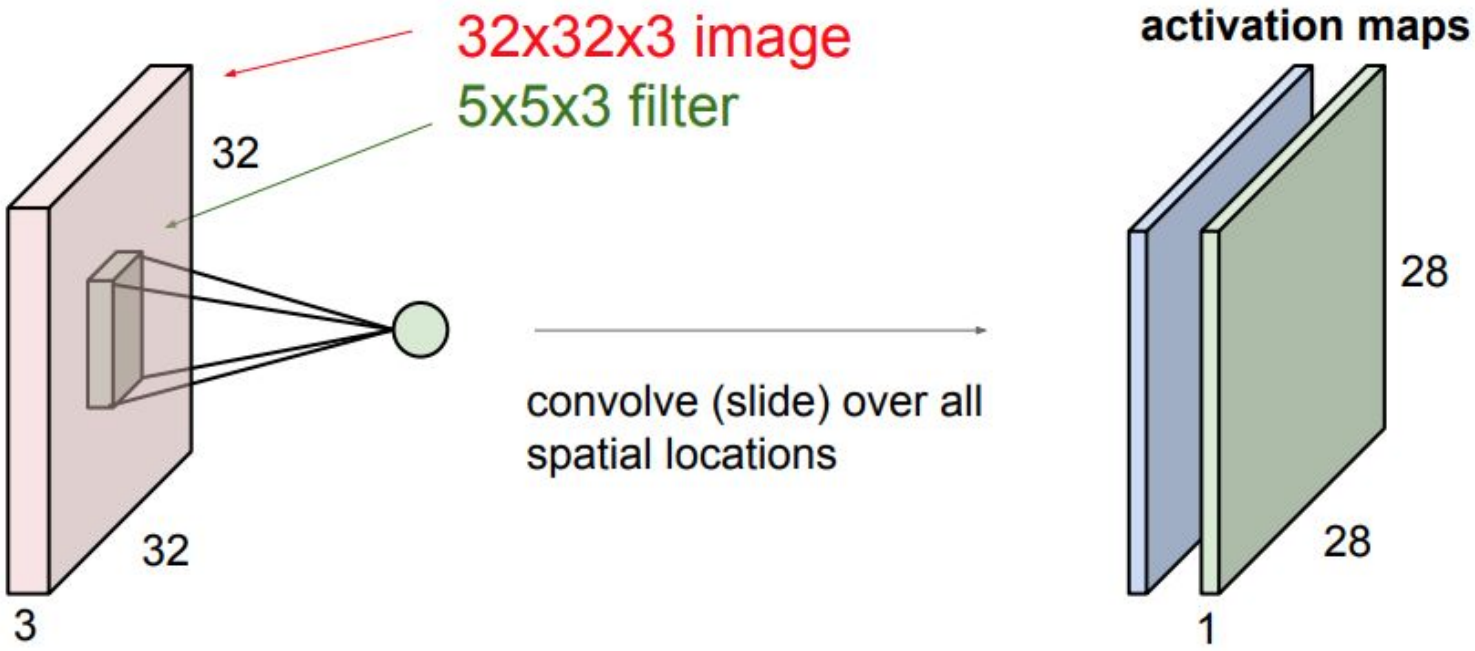
Convolution Layer



Andrej Karpathy, Bay Area Deep Learning School, 2016

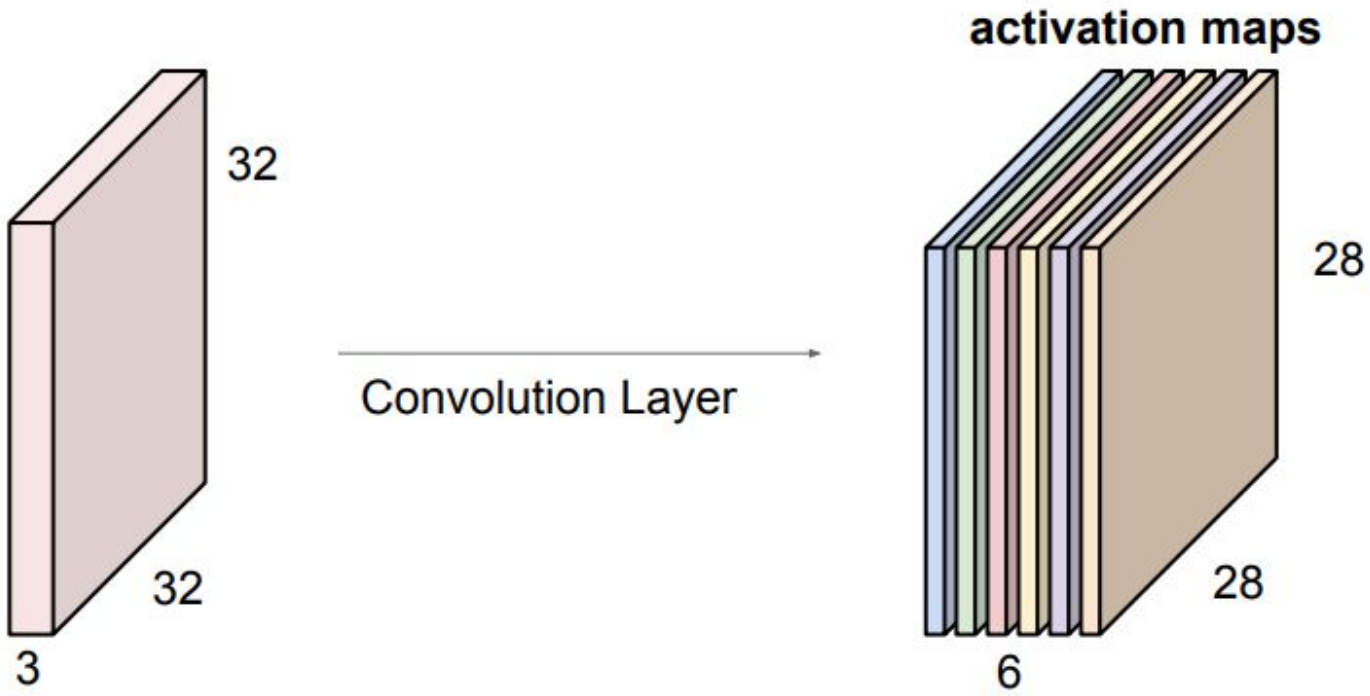
Convolution Layer

consider a second, green filter



Andrej Karpathy, Bay Area Deep Learning School, 2016

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

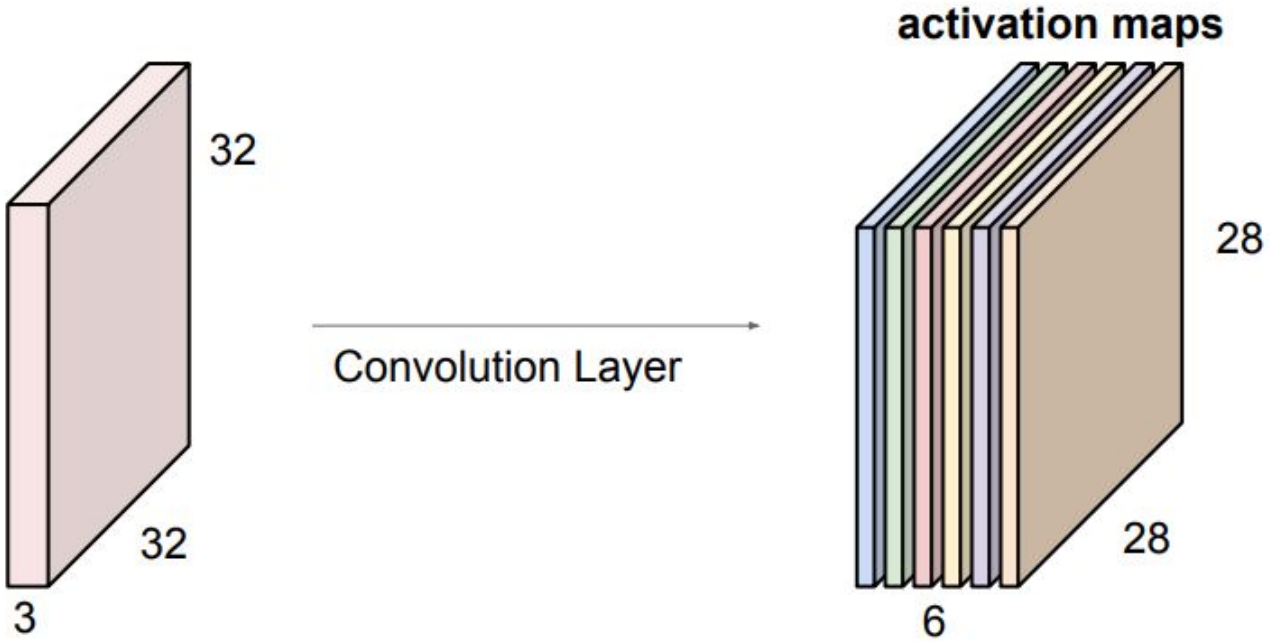


We stack these up to get a "new image" of size 28x28x6!

In pytorch, conv2 = nn.Conv2d(3, 6, kernel_size=5)

Andrej Karpathy, Bay Area Deep Learning School, 2016

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

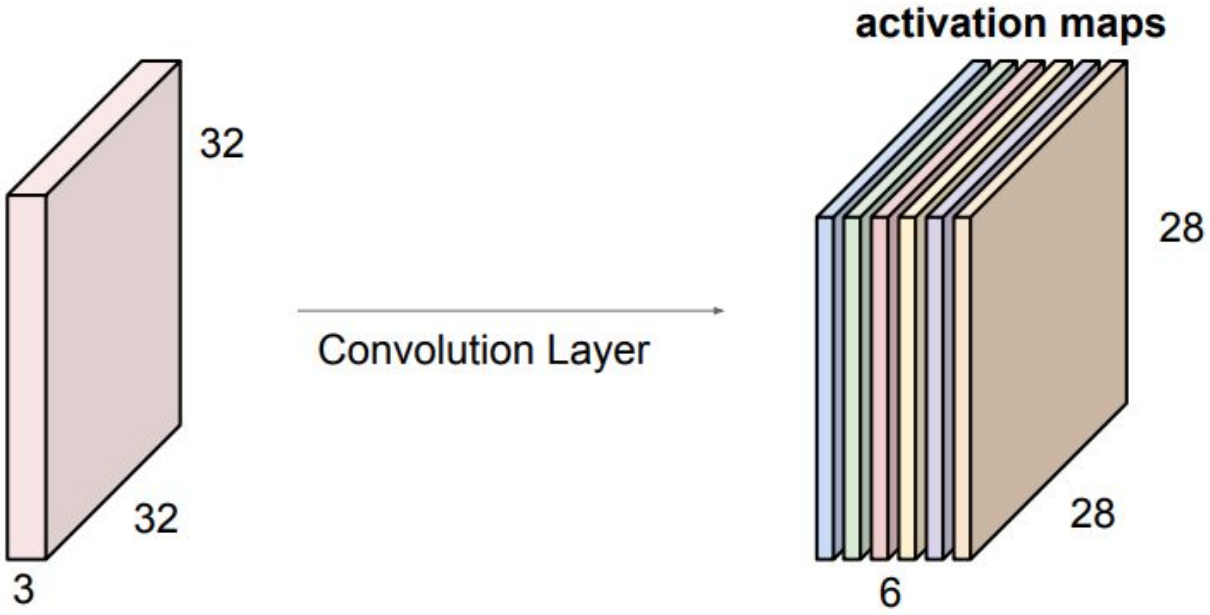


We processed [32x32x3] volume into [28x28x6] volume.

Q: how many parameters would this be if we used a fully connected layer instead?

Andrej Karpathy, Bay Area Deep Learning School, 2016

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



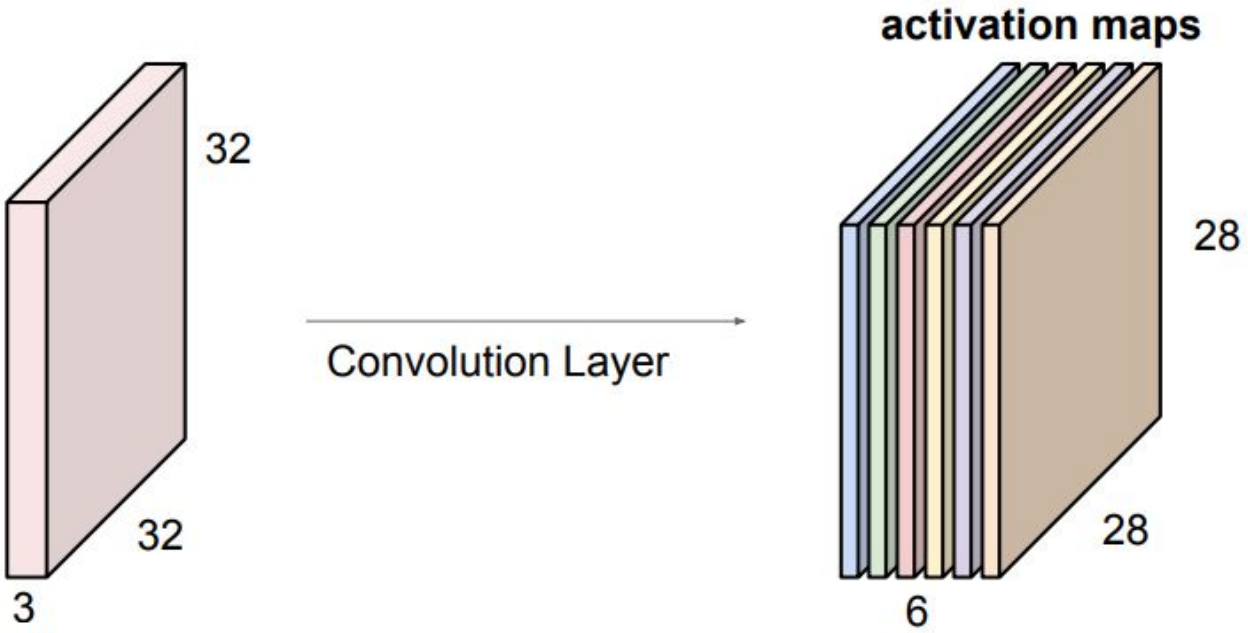
We processed [32x32x3] volume into [28x28x6] volume.

Q: how many parameters would this be if we used a fully connected layer instead?

A: (32*32*3)*(28*28*6) = **14.5M parameters**, **~14.5M multiplies**

Andrej Karpathy, Bay Area Deep Learning School, 2016

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

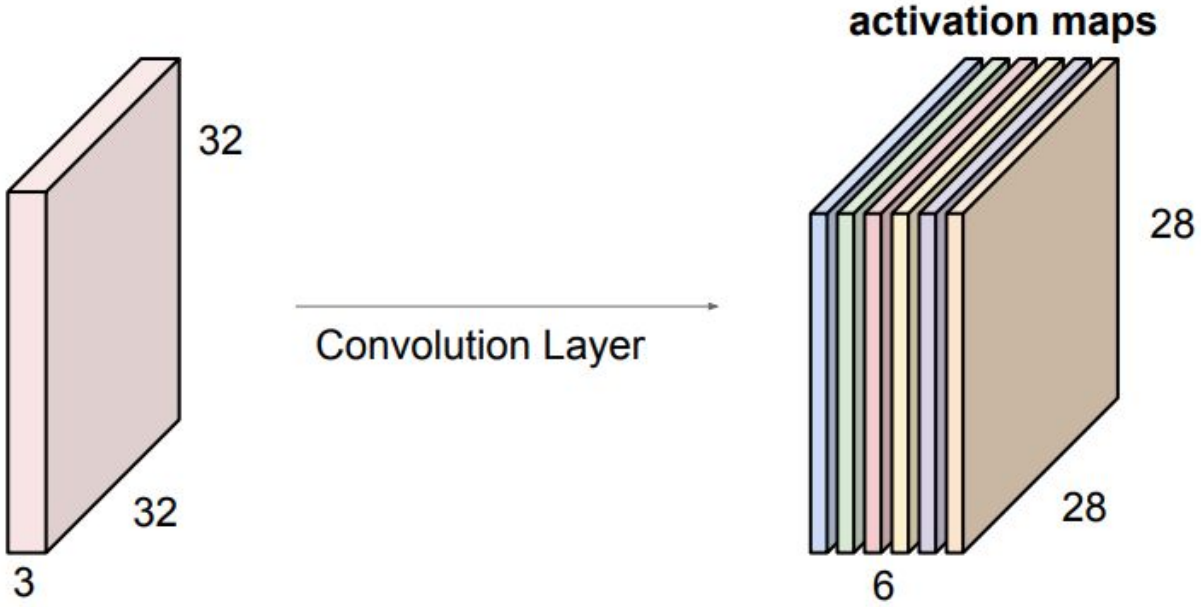


We processed [32x32x3] volume into [28x28x6] volume.

Q: how many parameters are used instead?

Andrej Karpathy, Bay Area Deep Learning School, 2016

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



We processed [32x32x3] volume into [28x28x6] volume.

Q: how many parameters are used instead?

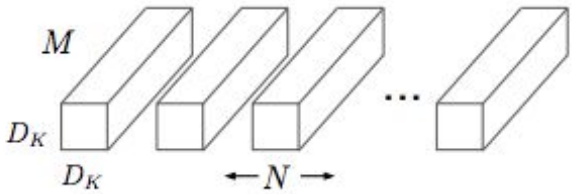
A: (5*5*3)*6 = **450 parameters**, (5*5*3)*(28*28*6) = **~350K multiplies**

Andrej Karpathy, Bay Area Deep Learning School, 2016

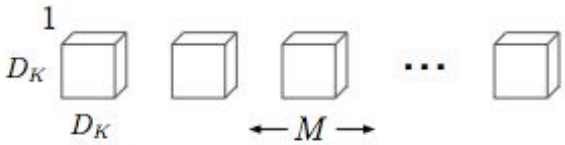
Convolution, Extended Work

Depthwise Convolution

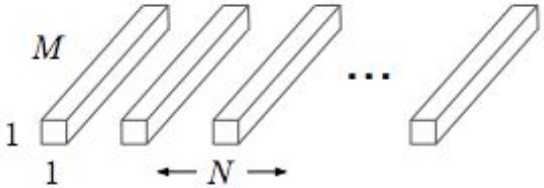
- Model compression



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

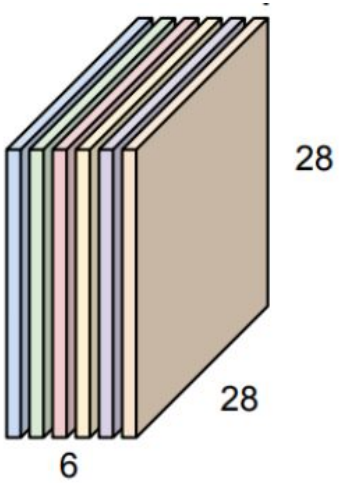


(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

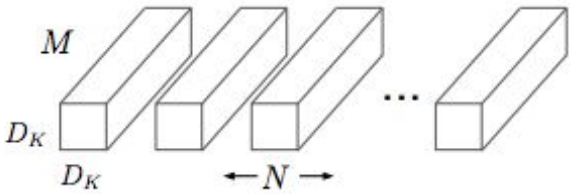
Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

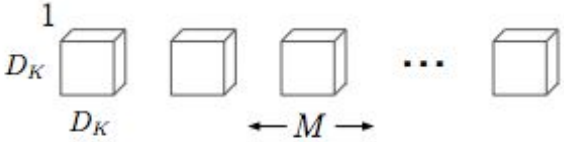
Convolution, Extended Work



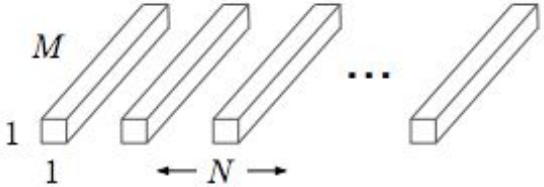
Input Feature Map



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Convolution, Extended Work

Deformable Convolution

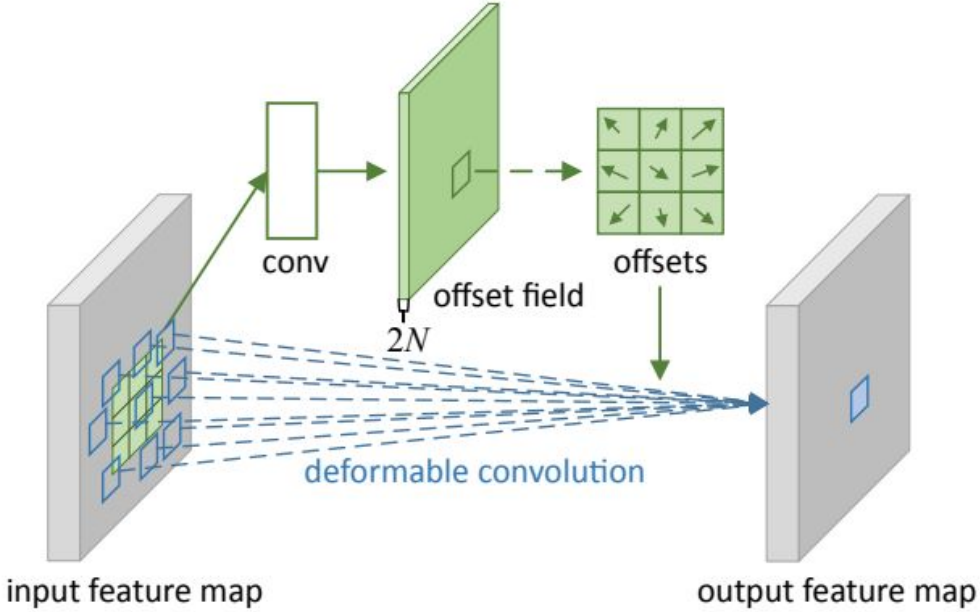
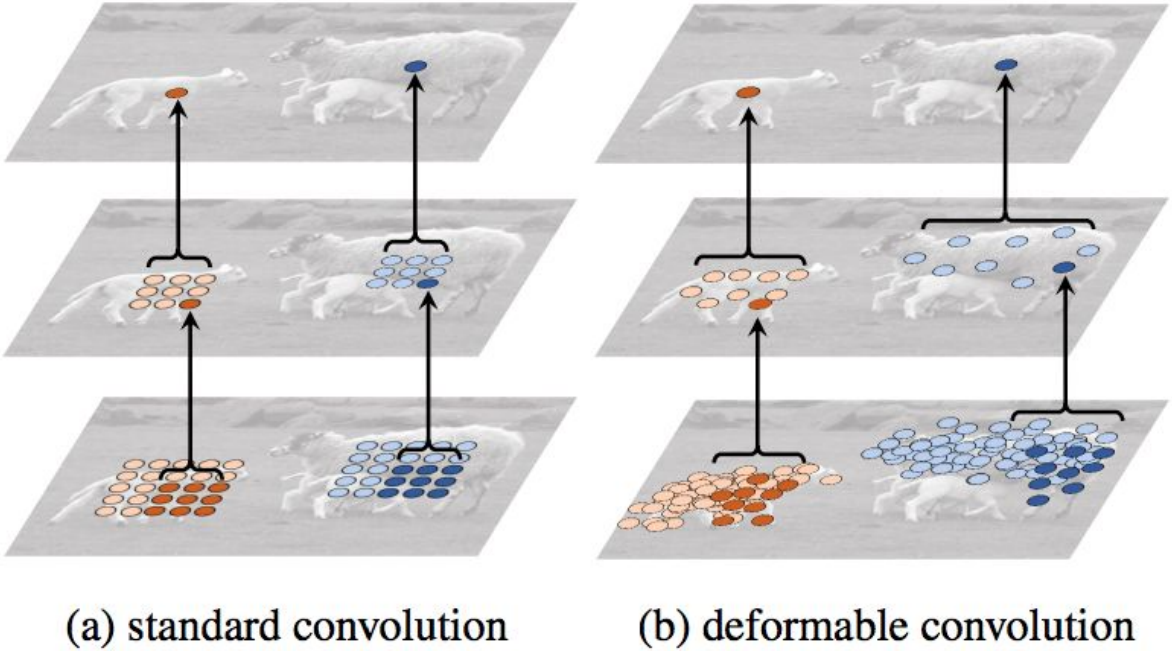


Figure 2: Illustration of 3×3 deformable convolution.

Deformable Convolutional Networks

Convolution, Extended Work

Deformable Convolution



(a) standard convolution (b) deformable convolution

Deformable Convolutional Networks

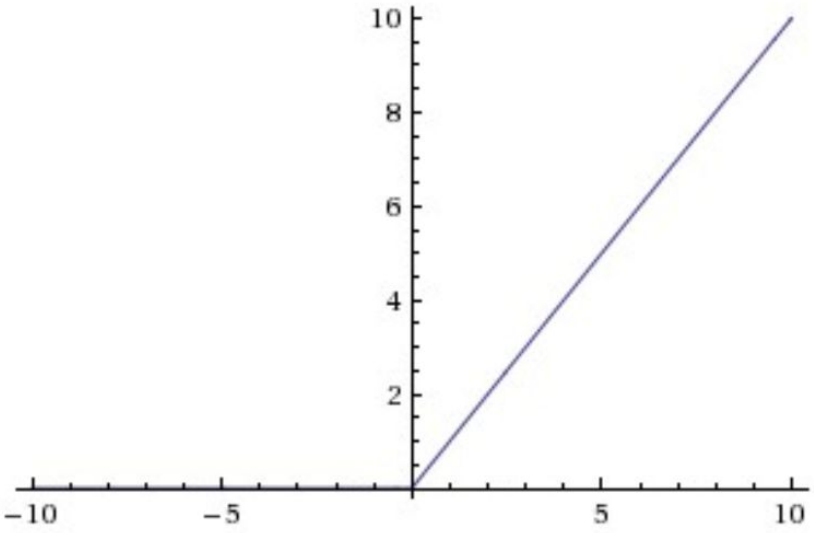
Non-Linear Function

Non-Linear Function

Conv -> ReLU -> Conv

Why we need Non-Linearity Function ?

- Conv, linear operation
- Two consecutive convolutional layers (no ReLU) are considered as one convolutional layer.



Rectified Linear Unit (ReLU)

Extended Work, PReLU

Parametric Rectified Linear Unit (PReLU)

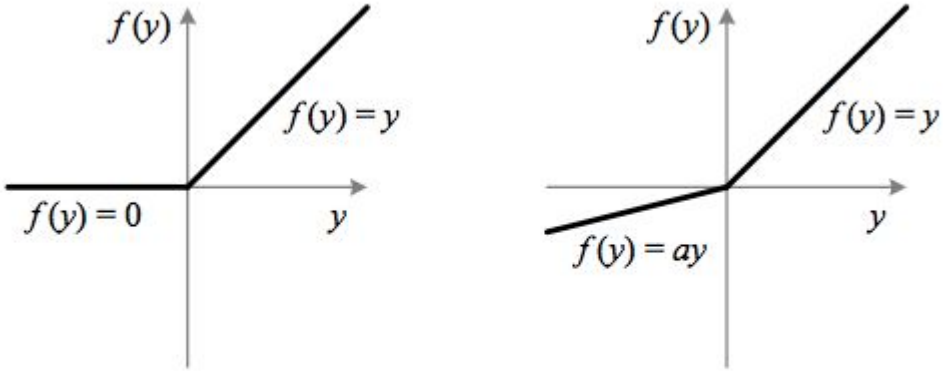
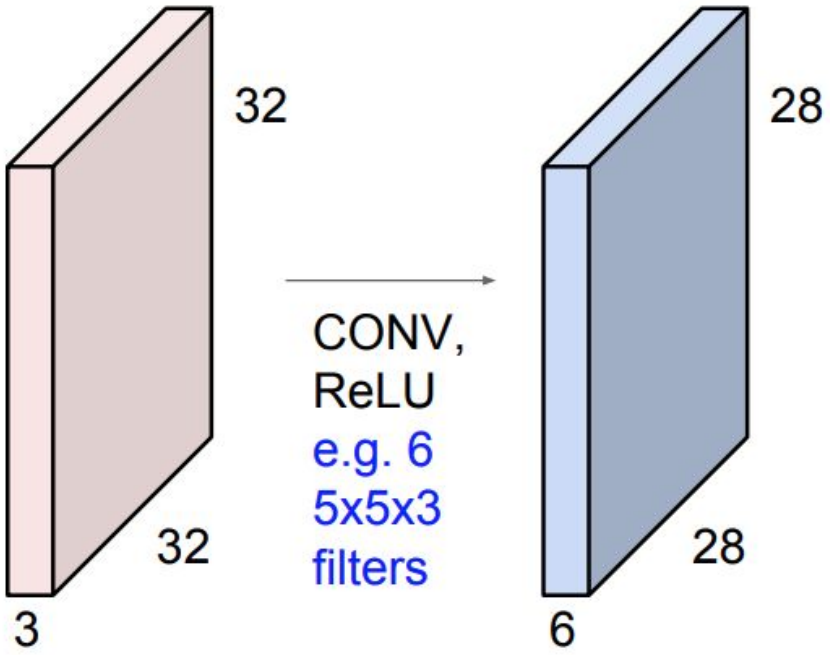


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

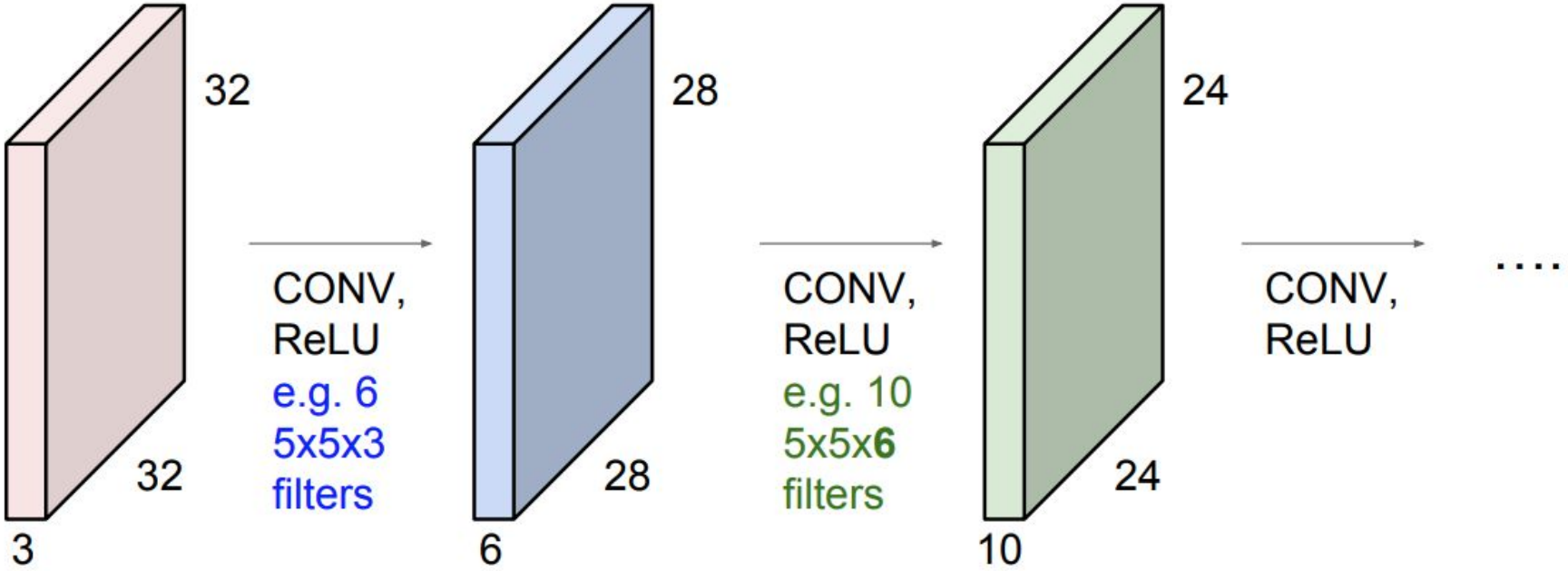
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

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



Andrej Karpathy, Bay Area Deep Learning School, 2016

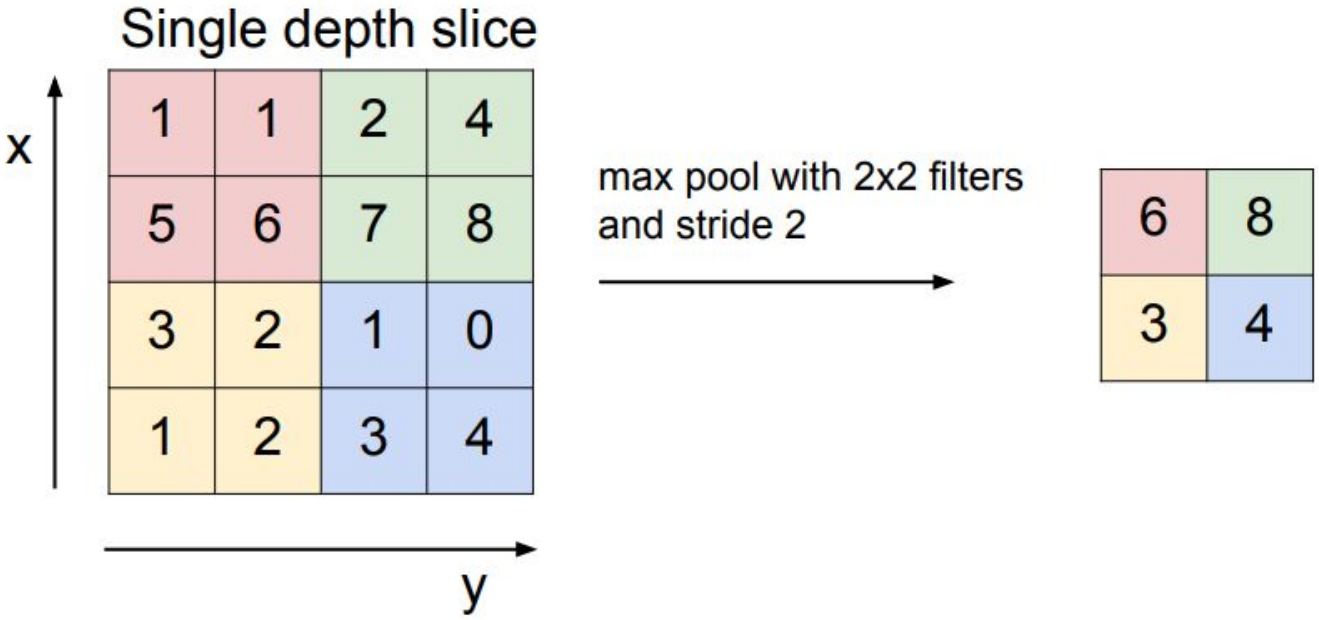
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Andrej Karpathy, Bay Area Deep Learning School, 2016

Pooling

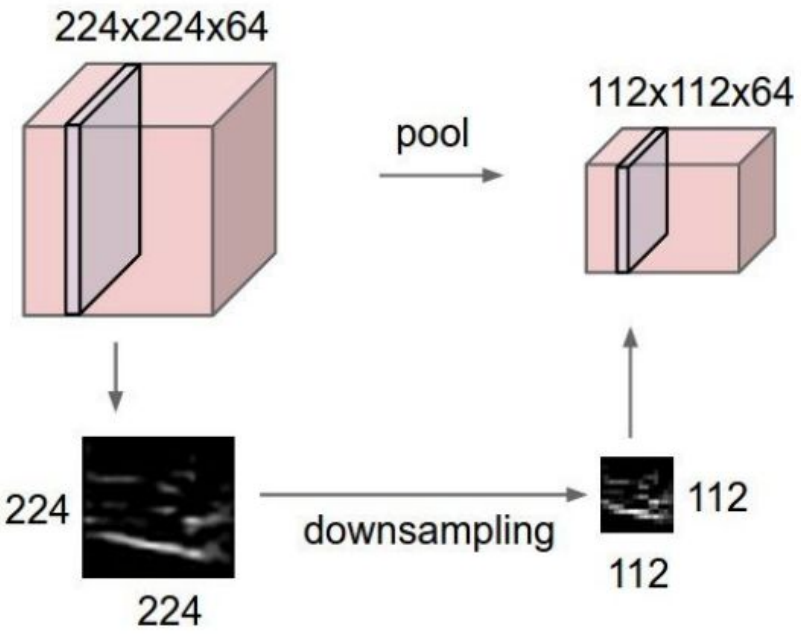
MAX POOLING



Andrej Karpathy, Bay Area Deep Learning School, 2016

Pooling layer

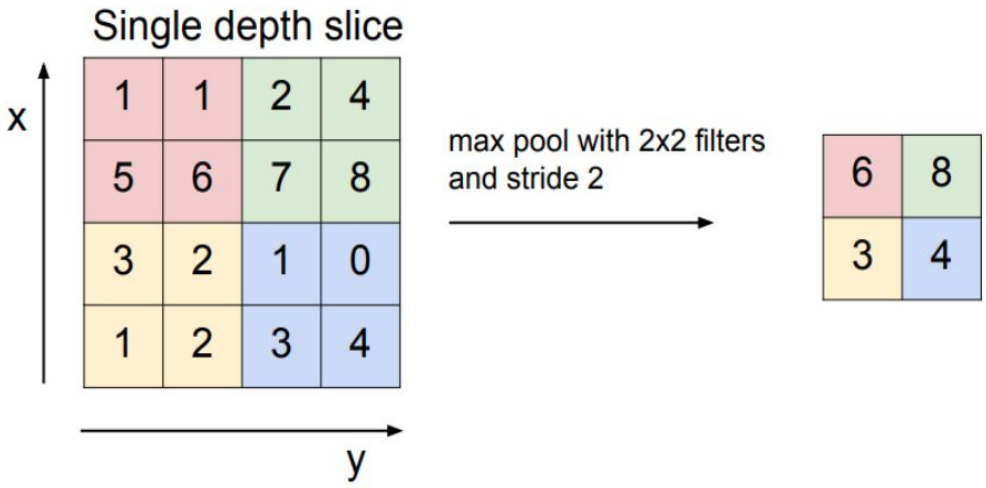
- makes the representations smaller and more manageable
- operates over each activation map independently:



Andrej Karpathy, Bay Area Deep Learning School, 2016

Pooling

- Max pooling
- Average pooling



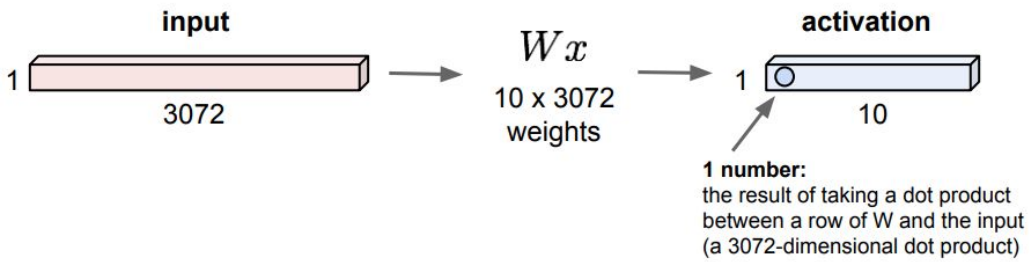
Fully Connected Layer

Fully Connected Layer

Representation for Feature Map:

(batch, channels, height, width)

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer:

(batch, channels, 1, 1)
or
(batch, channels)

#parameters: input_channel * output_channel

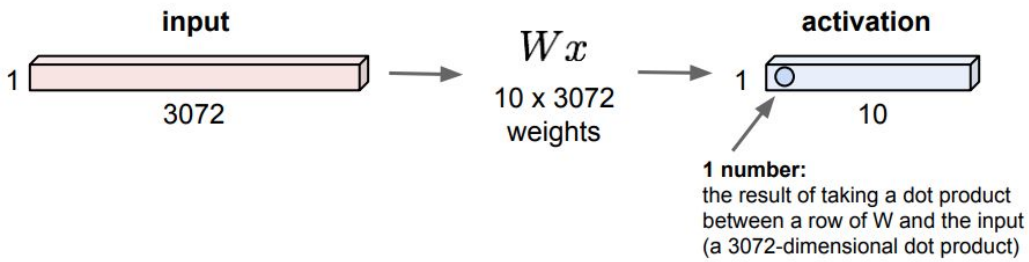
Stanford cs231n.

Fully Connected Layer

Usually we apply reshape operation for input feature map

(n, c, h, w) -> (n, c*h*w, 1, 1)
or (n, c*h*w)

32x32x3 image -> stretch to 3072 x 1



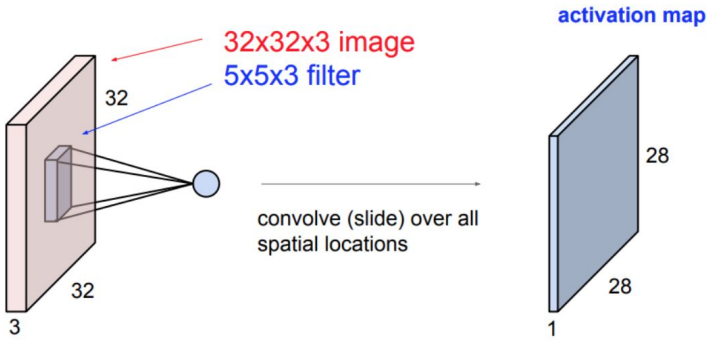
#parameters: input_channel * output_channel

Stanford cs231n.

Convolutional Vs. Fully Connected Layer

Convolutional Layer

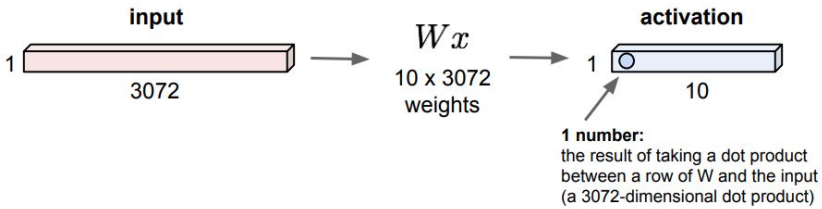
- Local information
- Kernels are shared



Fully Connected Layer

- Global information
- #parameters large

32x32x3 image -> stretch to 3072 x 1



Stanford cs231n.

Agenda

1. Forward

   Conv, Fully Connected, Pooing, non-linear Function
   Loss functions

2. Back Propagation, Computing Gradient

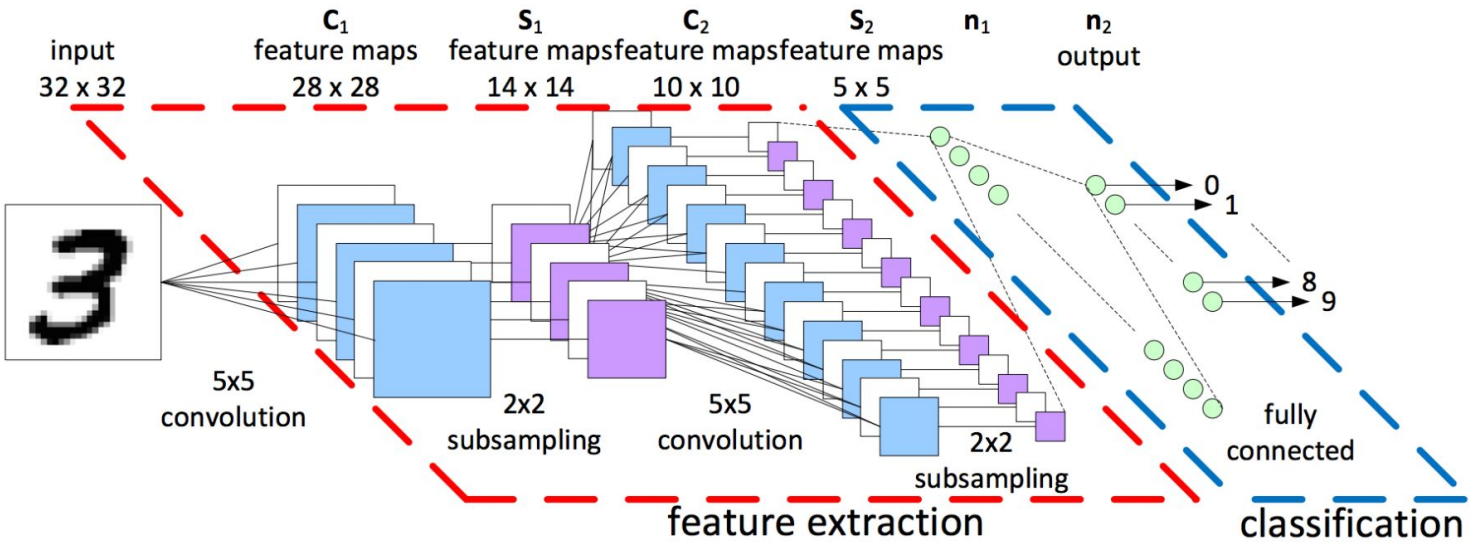
   Chain rule

3. Updating Parameters

   SGD

4. Training

Overview



1. **Forward**: compute output of each layer
2. **Back propagation**: compute gradient
3. **Updating**: update the parameters with computed gradient

Loss Functions

- Softmax
- Regression
- Contrastive/Triplet Loss

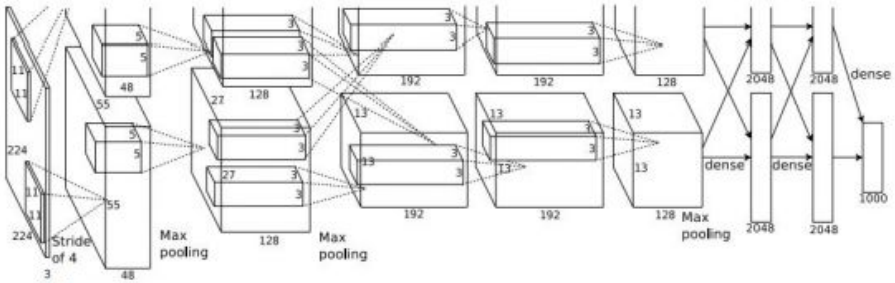
Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Stanford cs231n.

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

cat	**3.2**
car	5.1
frog	-1.7

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

where

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

cat	**3.2**
car	5.1
frog	-1.7

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad s = f(x_i; W)$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

y_i is the groundtruth

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

---

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

y_i is the groundtruth

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

cat	**3.2**	exp →	**24.5**
car	5.1		164.0
frog	-1.7		0.18

unnormalized log probabilities

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

Groundtruth
(one hot vector)

[1,
0,
0]

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

→ L_i = -log(0.13)
= **0.89**

Groundtruth
(one hot vector)

[1,
0,
0]

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

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

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

unnormalized probabilities

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

→ L_i = -log(0.13)
= **0.89**

Stanford cs231n.

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

Q2: Usually at initialization W is small so all s ≈ 0.
What is the loss?

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

→ L_i = -log(0.13)
= **0.89**

Stanford cs231n.

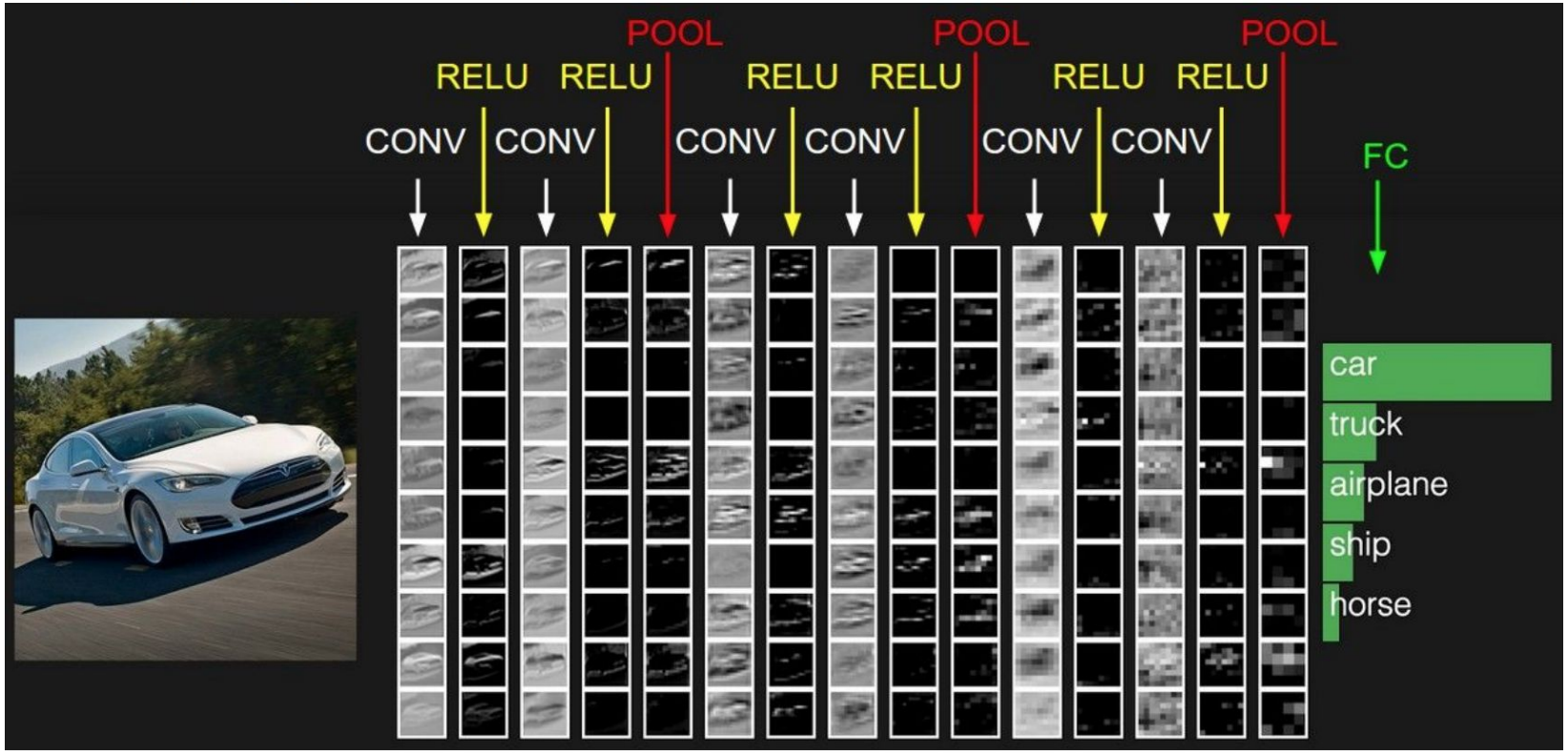
Loss Functions

- Softmax
- Regression

  Bounding box regression
- Contrastive/Triplelet Loss

  Object(face) recognition, clustering

two more layers to go: POOL/FC



Andrej Karpathy, Bay Area Deep Learning School, 2016

Define Network

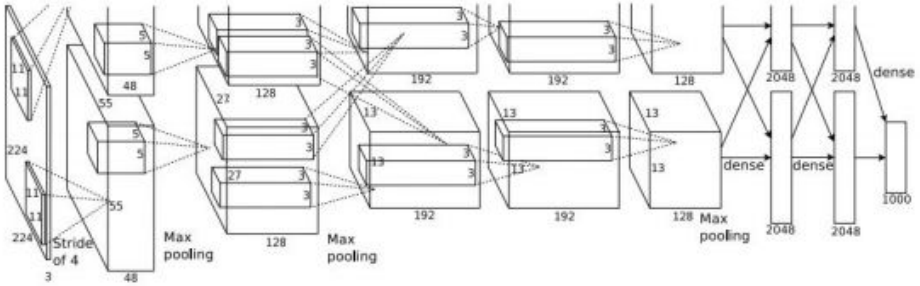
```
52  class Net(nn.Module):
53      def __init__(self):
54          super(Net, self).__init__()
55          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
56          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
57          self.conv2_drop = nn.Dropout2d()
58          self.fc1 = nn.Linear(320, 50)
59          self.fc2 = nn.Linear(50, 10)
60
61      def forward(self, x):
62          x = F.relu(F.max_pool2d(self.conv1(x), 2))
63          x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
64          x = x.view(-1, 320)
65          x = F.relu(self.fc1(x))
66          x = F.dropout(x, training=self.training)
67          x = self.fc2(x)
68          return F.log_softmax(x)
69
70  model = Net()
71  if args.cuda:
72      model.cuda()
```

https://goo.gl/mQEw15

Network Case Study (optional)

Case Study: AlexNet

[Krizhevsky et al. 2012]

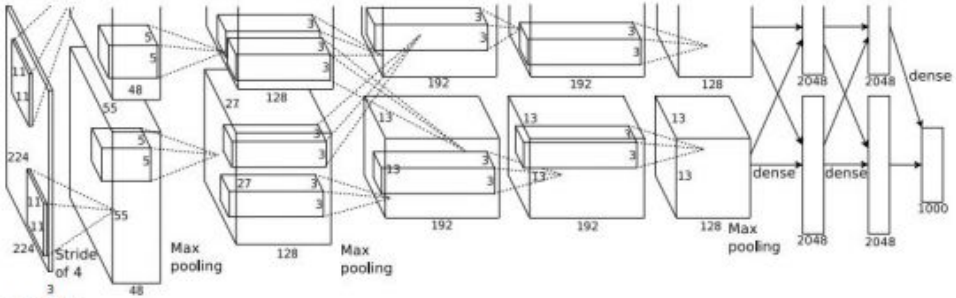


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

VGG Network

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
conv3-64	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	**conv3-256**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Andrej Karpathy, Bay Area Deep Learning School, 2016

VGG Network

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Andrej Karpathy, Bay Area Deep Learning School, 2016

Agenda

1. Forward

   Conv, Fully Connected, Pooing, non-linear Function
   Loss functions

2. Back Propagation, Computing Gradient

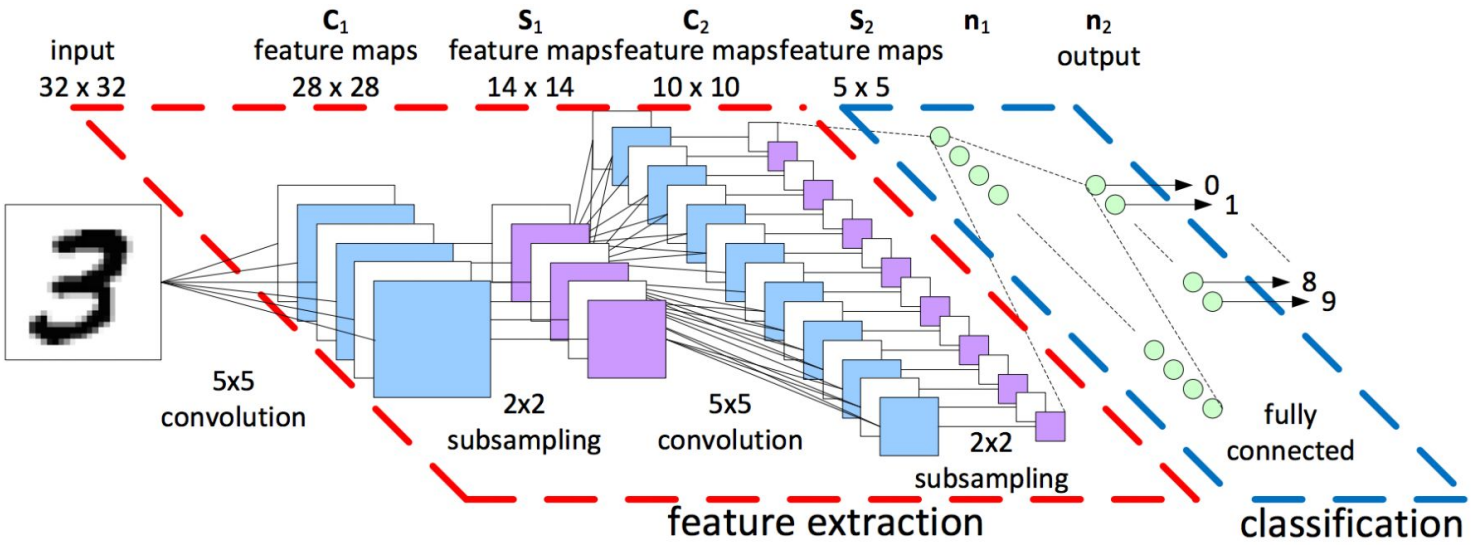
   Chain rule

3. Updating Parameters

   SGD

4. Training

Overview



1. **Forward**: compute output of each layer
2. **Backward**: compute gradient
3. **Update**: update the parameters with computed gradient

Back Propagation

Parameters:

Convolutional Layer, Fully Connected Layer

PReLU, Batch Normalization

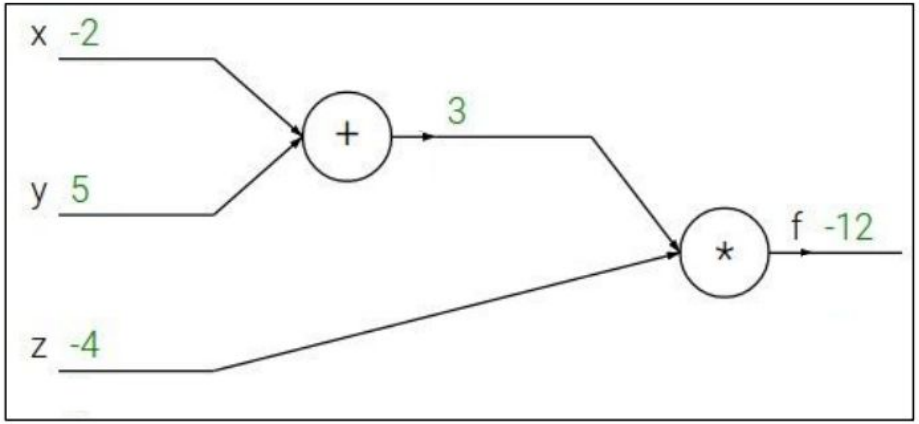
No Parameters:

Pooling, ReLU, Dropout

Backpropagation: a simple example

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

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



Stanford cs231n.

Backpropagation: a simple example

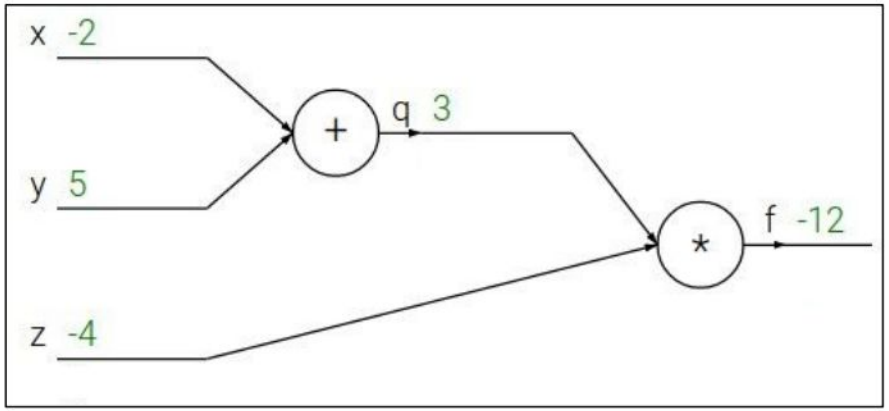
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

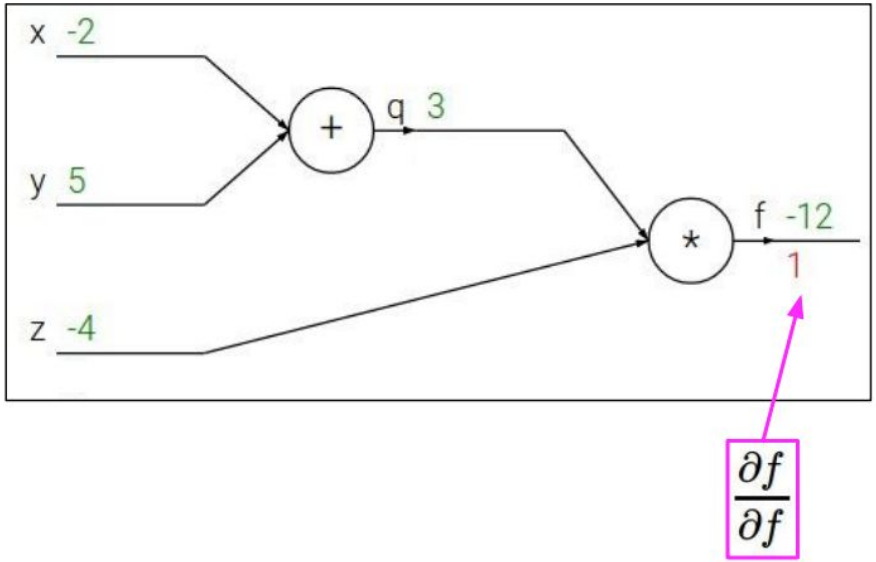
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

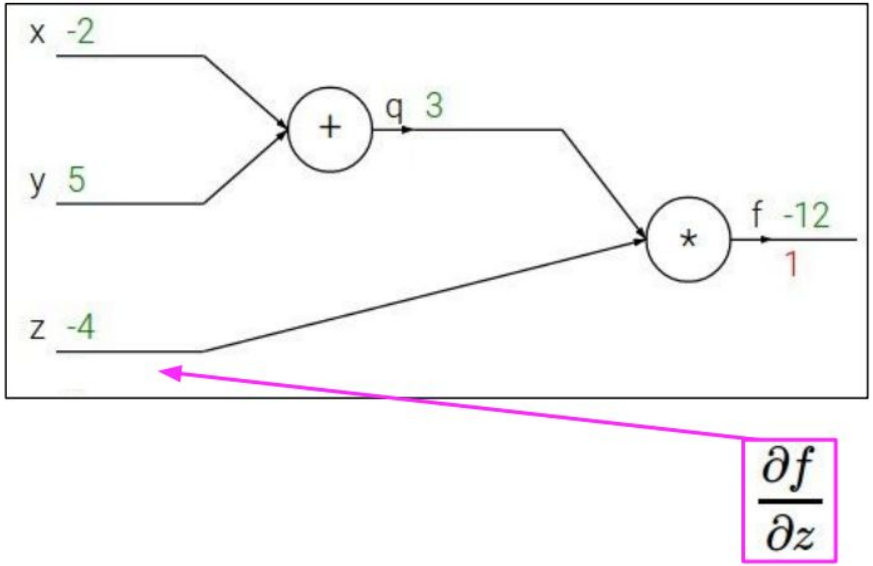
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

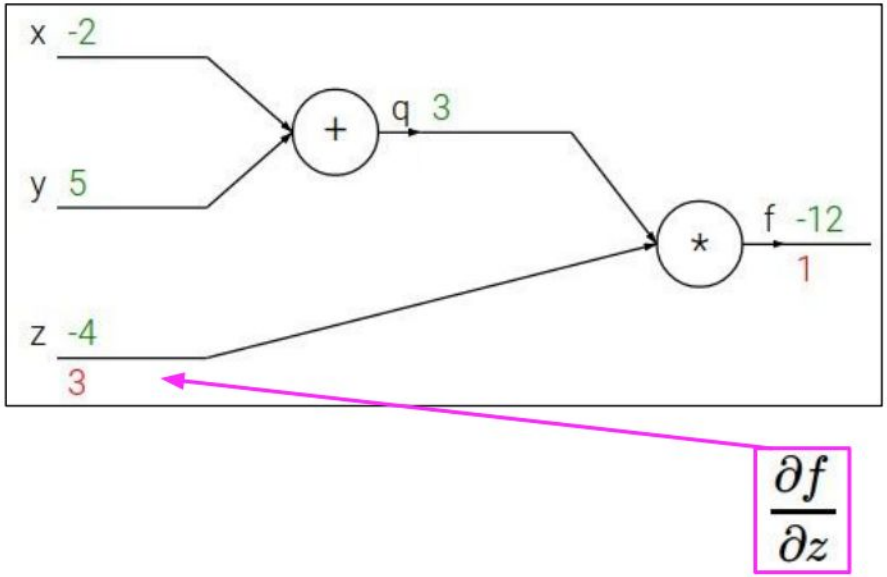
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

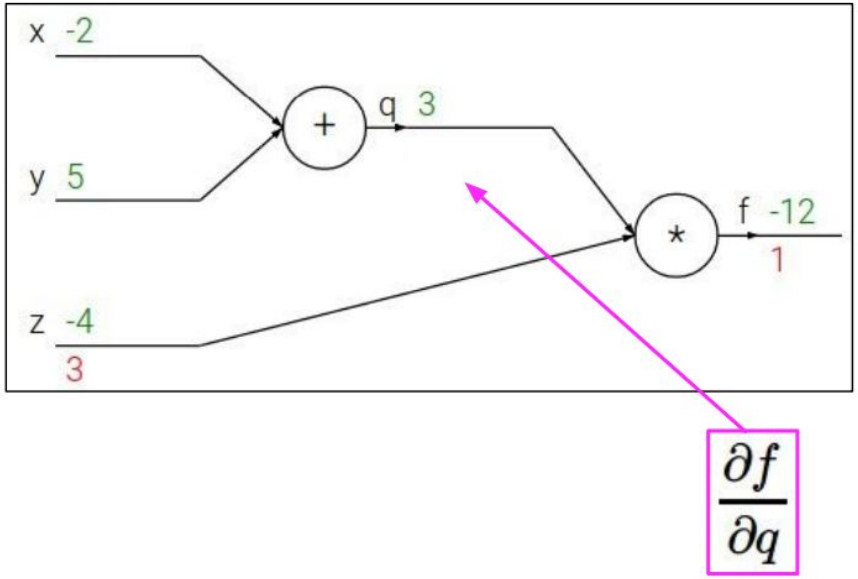
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

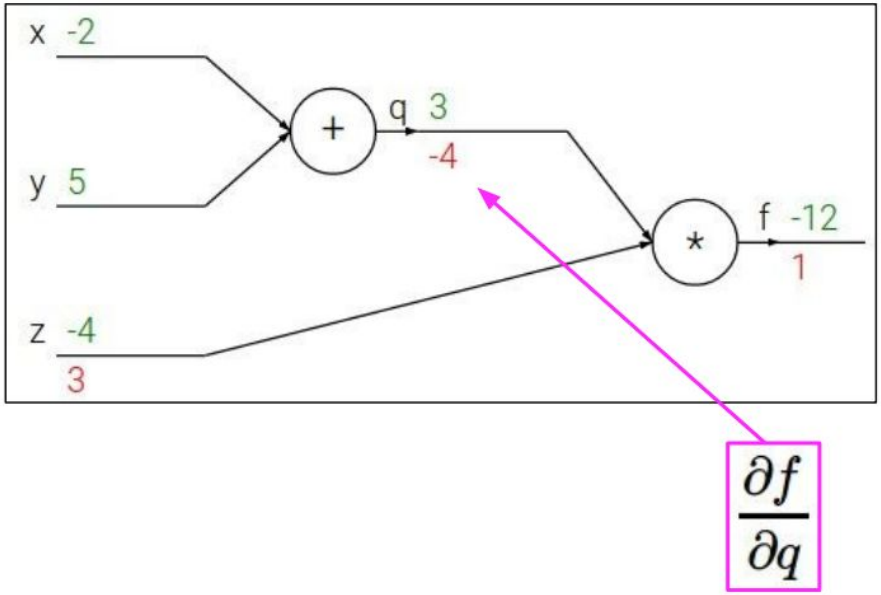
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Stanford cs231n.

Backpropagation: a simple example

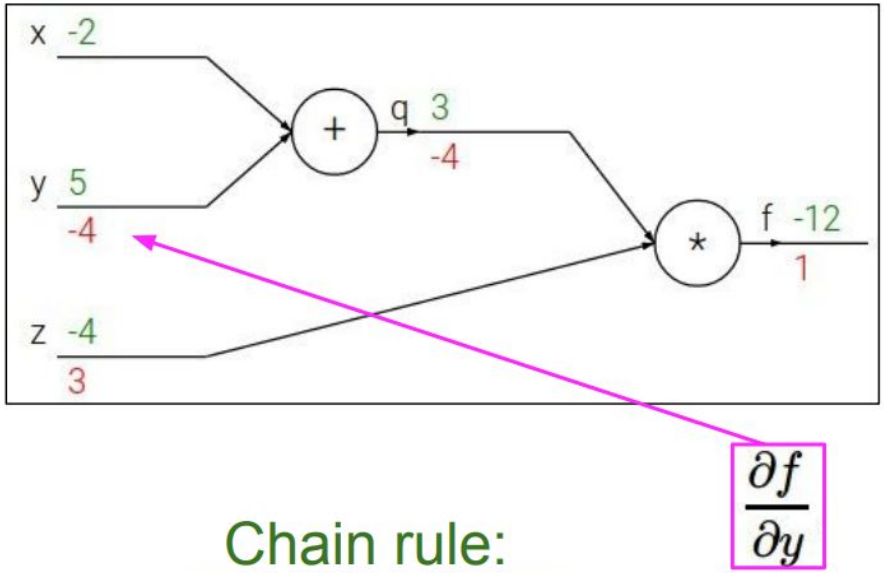
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

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



Chain rule:

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

Stanford cs231n.

Backpropagation: a simple example

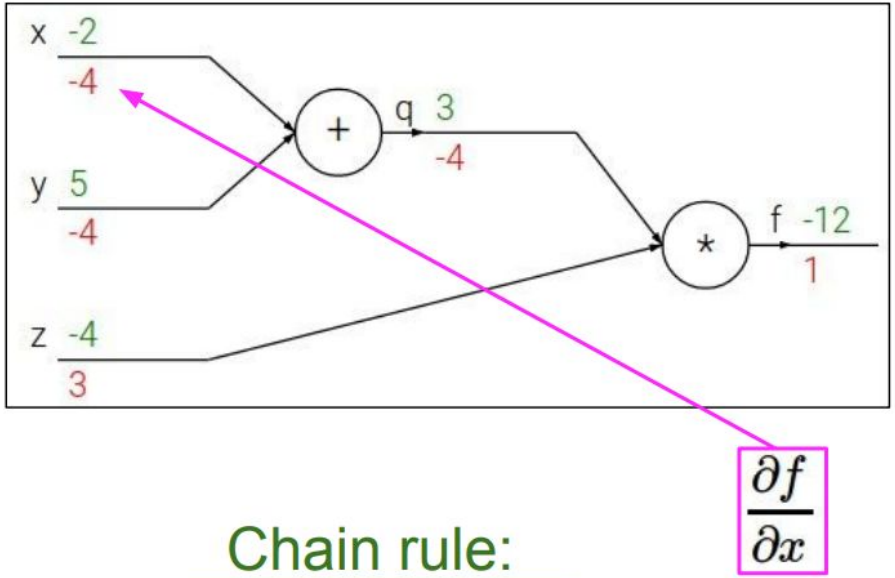
$$f(x, y, z) = (x + y)z$$

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

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

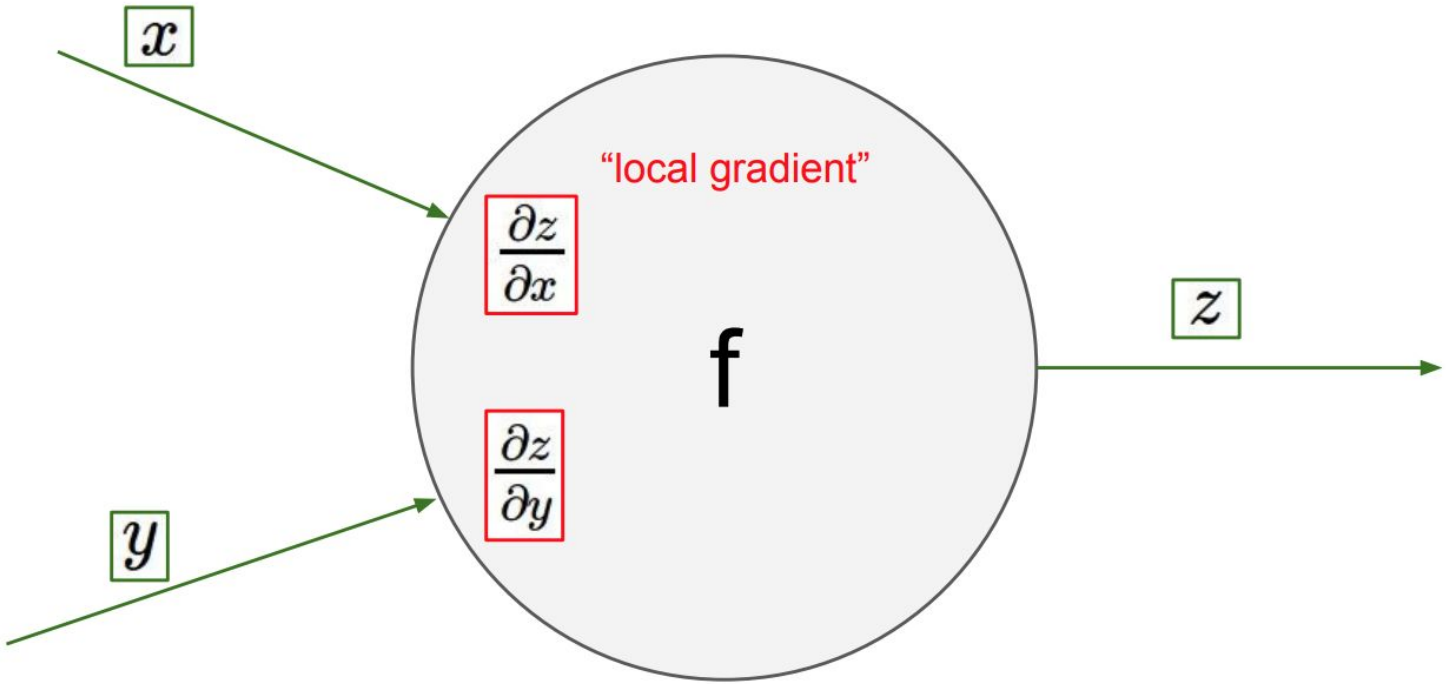
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



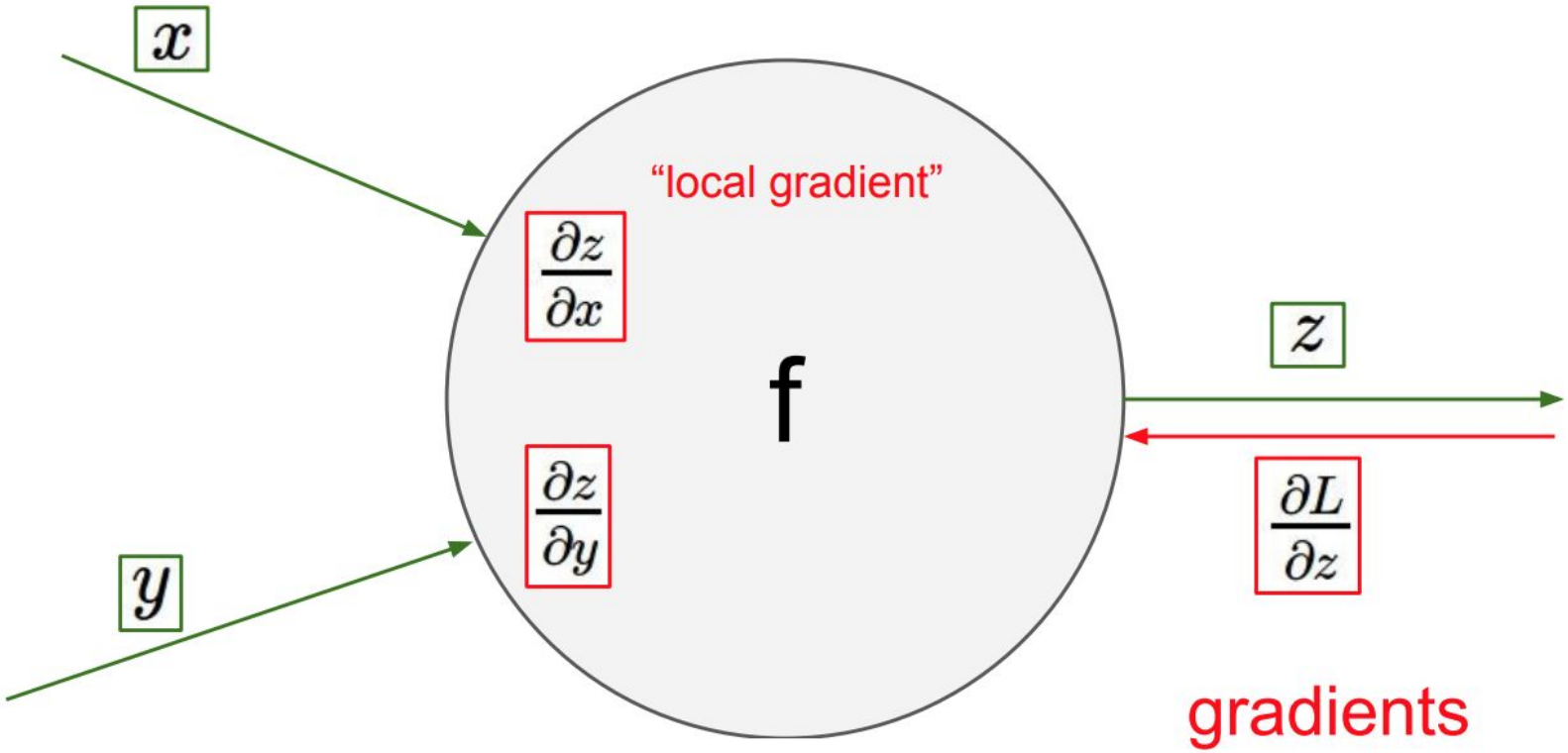
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

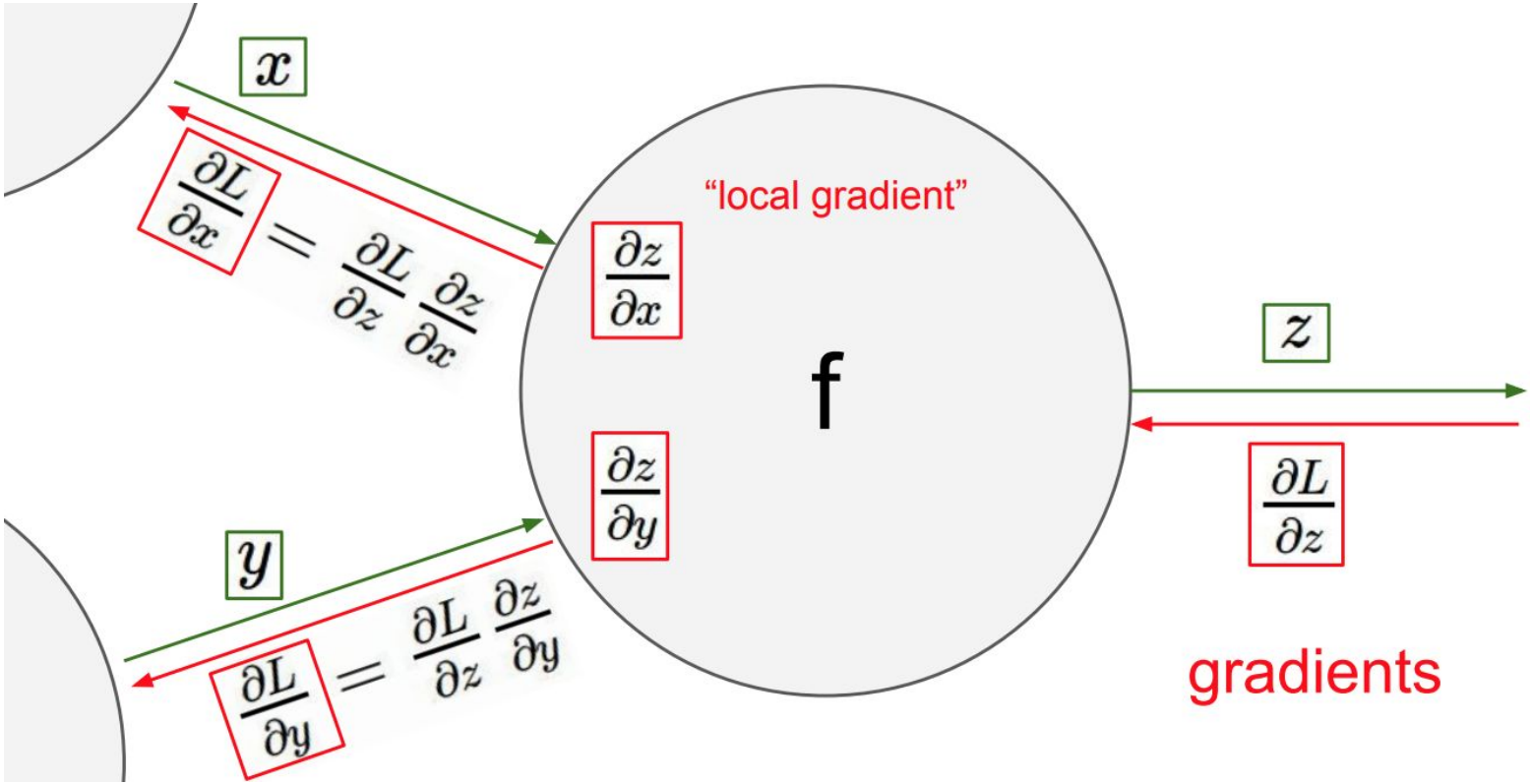
Stanford cs231n.



Stanford cs231n.



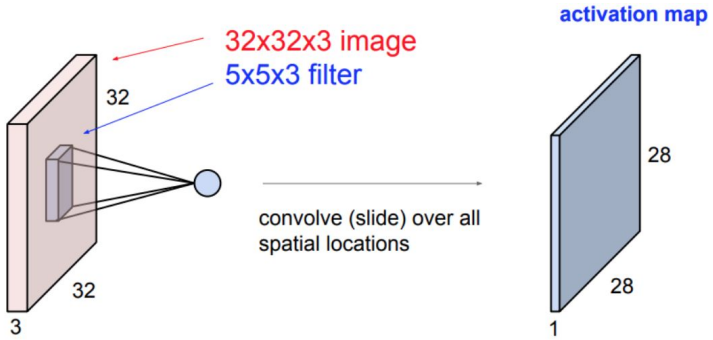
Stanford cs231n.



Stanford cs231n.

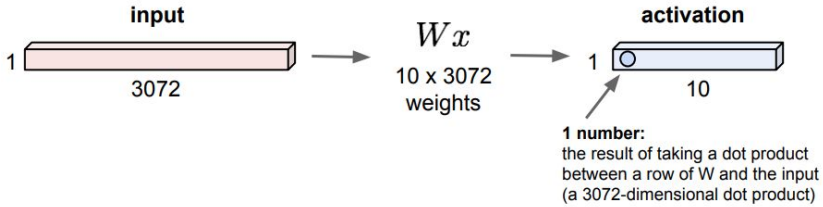
Back Propagation

Convolutional Layer



Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Back Propagation

PyTorch: Autograd

A PyTorch **Variable** is a node in a computational graph

x.data is a Tensor

x.grad is a Variable of gradients (same shape as x.data)

x.grad.data is a Tensor of gradients

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in), requires_grad=False)
y = Variable(torch.randn(N, D_out), requires_grad=False)
w1 = Variable(torch.randn(D_in, H), requires_grad=True)
w2 = Variable(torch.randn(H, D_out), requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    if w1.grad: w1.grad.data.zero_()
    if w2.grad: w2.grad.data.zero_()
    loss.backward()

    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data
```

Once you finish your computation you can call .backward() and have all the gradients computed automatically. “

Stanford cs231n. *Pytorch Tutorial. www.pytorch.org*

Agenda

1. Forward

   Conv, Fully Connected, Pooing, non-linear Function
   Loss functions

2. Backward, Computing Gradient

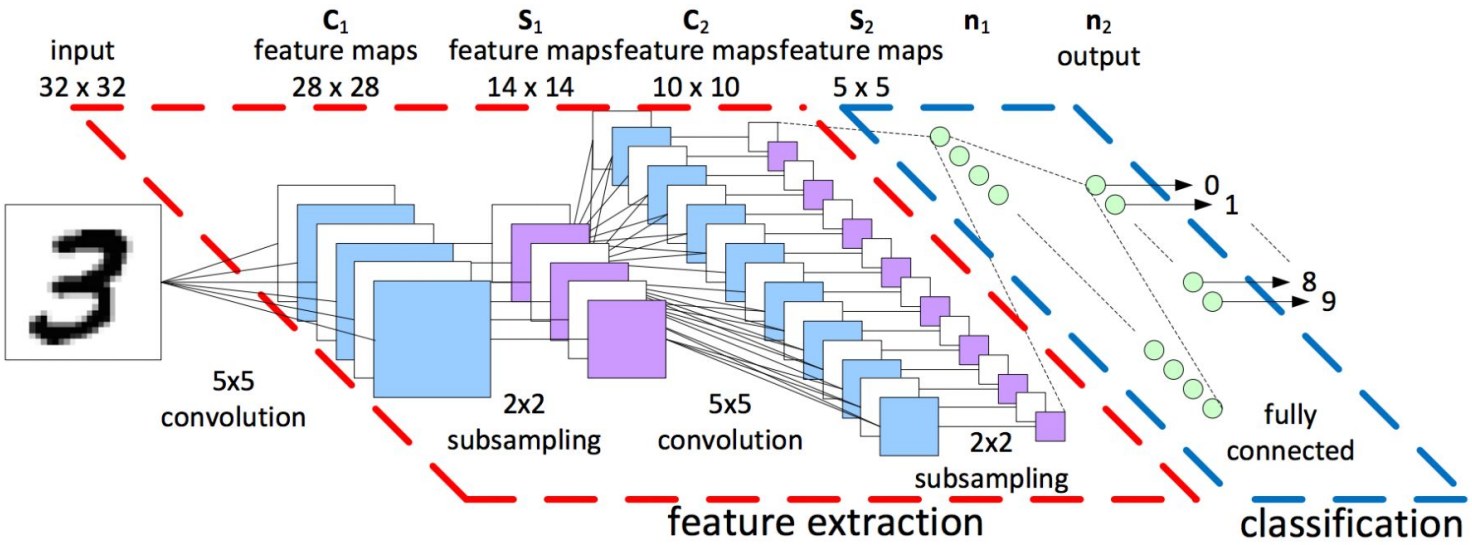
   Chain rule

3. Updating Parameters

   SGD

4. Training

Overview



1. **Forward**: compute output of each layer
2. **Backward**: compute gradient
3. **Update**: update the parameters with computed gradient

Updating Gradient

Vanilla update. The simplest form of update is to change the parameters along the negative gradient direction (since the gradient indicates the direction of increase, but we usually wish to minimize a loss function). Assuming a vector of parameters `x` and the gradient `dx`, the simplest update has the form:

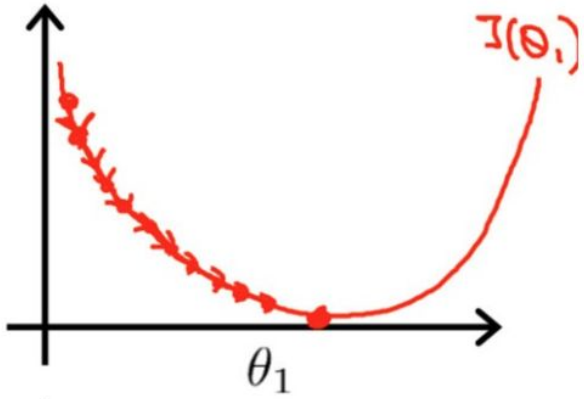
```
# Vanilla update
x += - learning_rate * dx
```

Stanford cs231n.

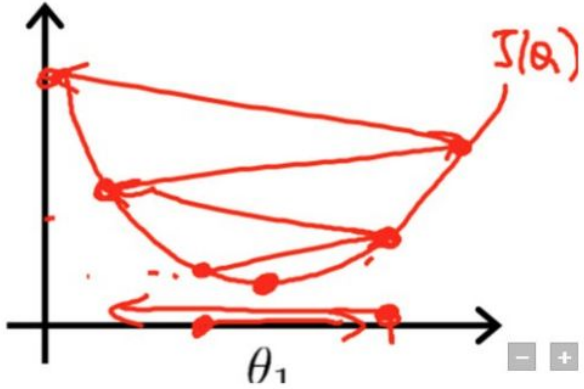
Updating Gradient

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



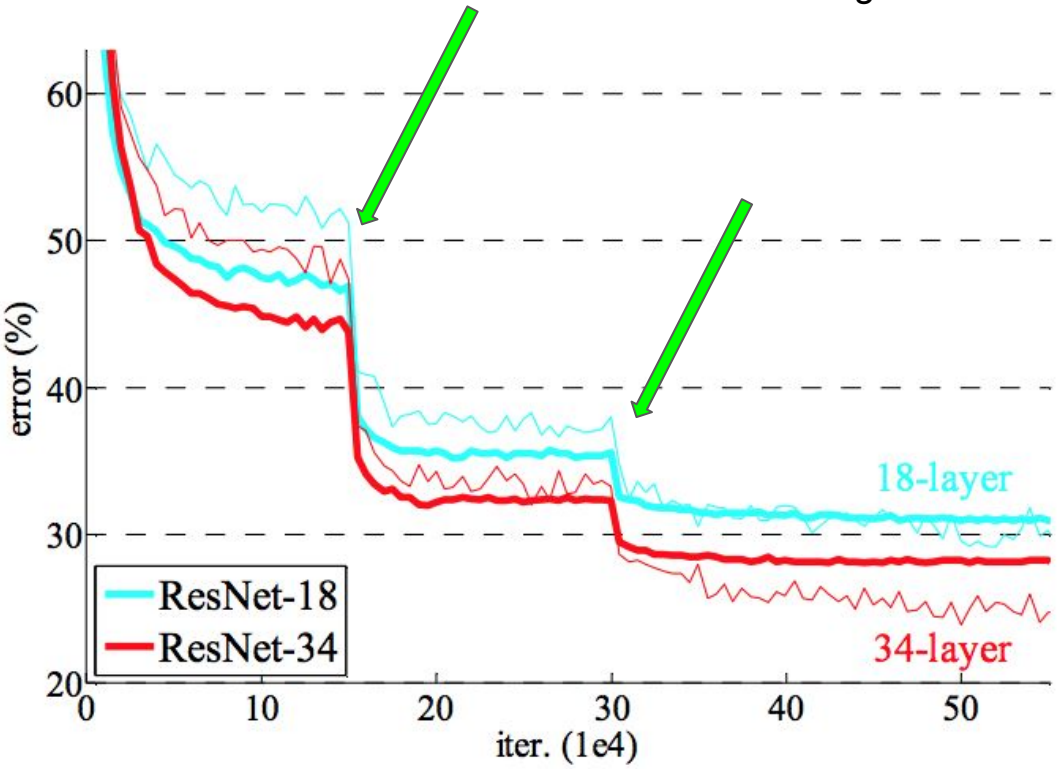
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Andrew Ng's Machine Learning course, Coursera

Updating Gradient

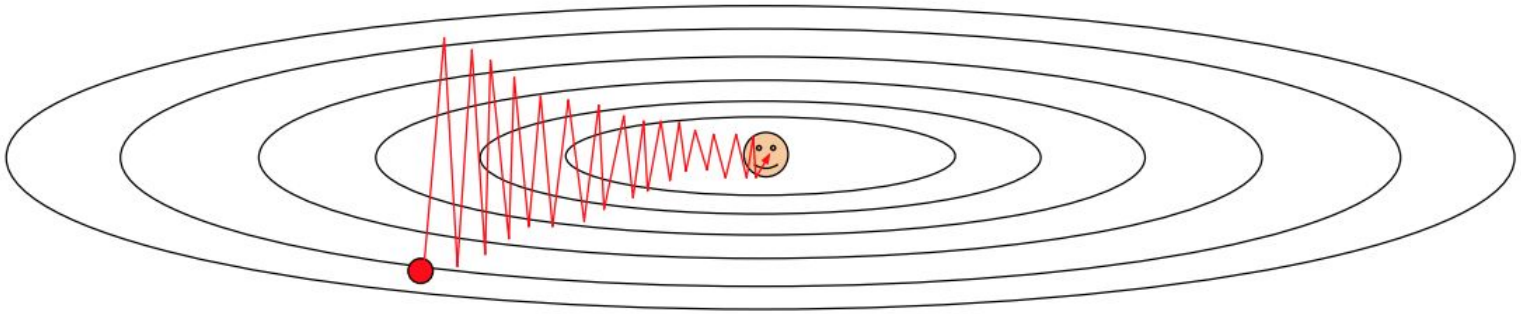
decrease the learning rate after several epochs



Deep Residual Learning for Image Recognition

Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



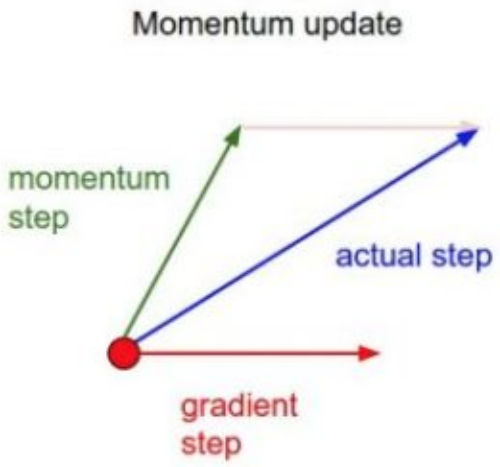
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Stanford cs231n.

Updating Gradient

```
# Vanilla update
x += - learning_rate * dx
```

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```



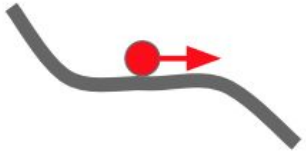
Momentum

Stanford cs231n.

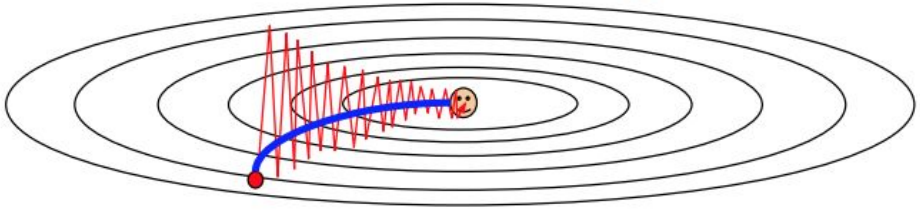
SGD + Momentum

Local Minima

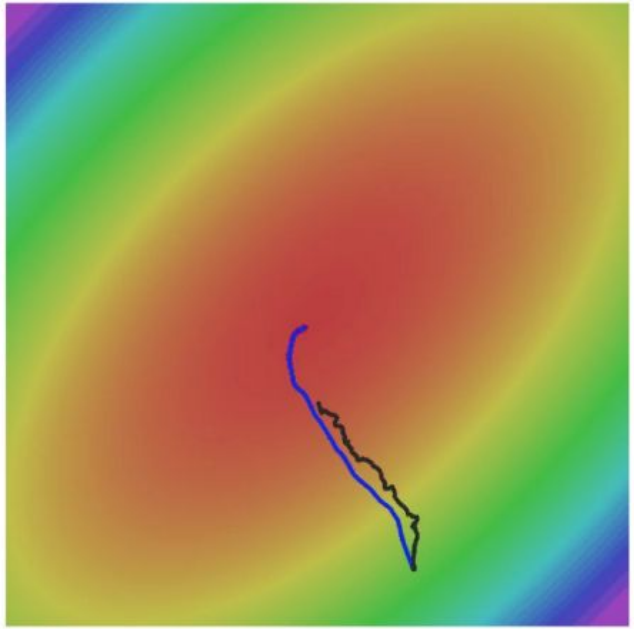
Saddle points



Poor Conditioning



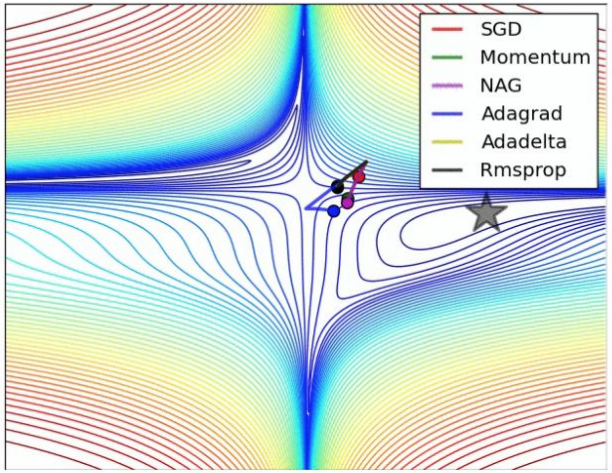
Gradient Noise

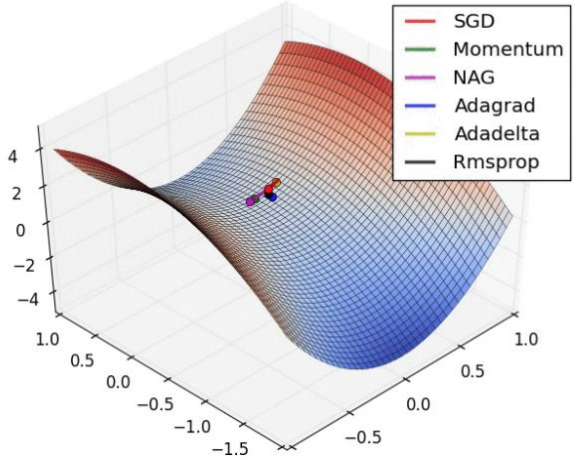


Stanford cs231n.

Other updating Strategies

- **SGD**
- **Momentum**
- Rmsprop
- Adagrad
- **Adam**



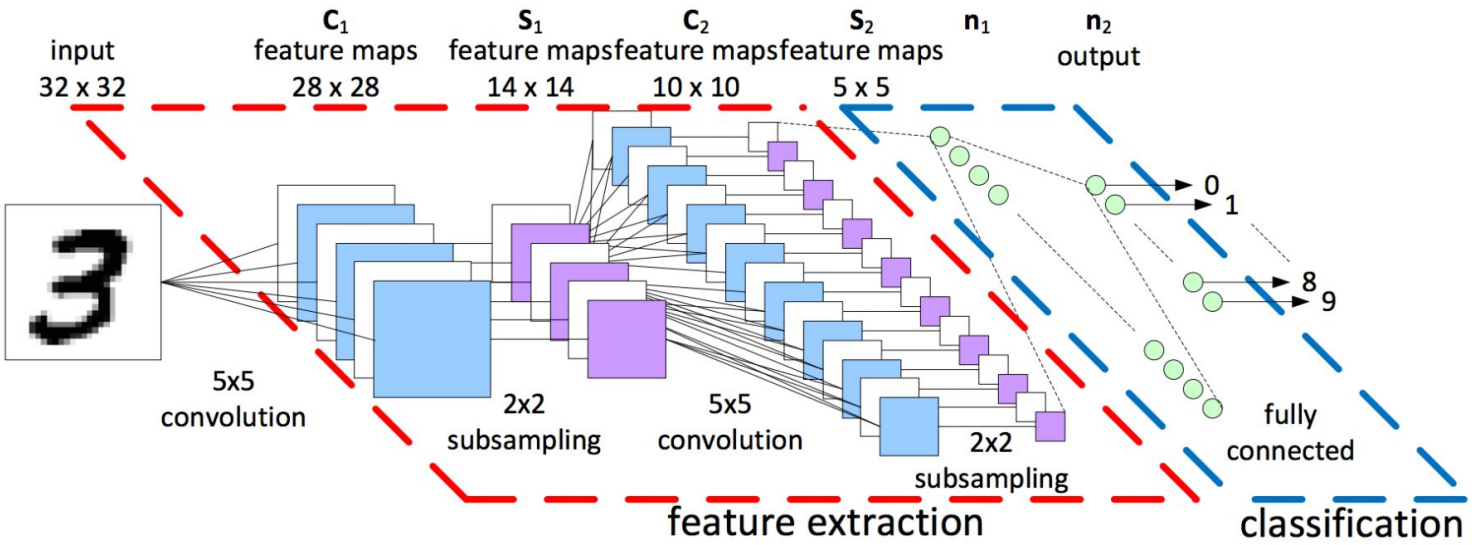


Animations that may help your intuitions about the learning process dynamics. Left: Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. Right: A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed. Images credit: Alec Radford.

Gif Animation, http://cs231n.github.io/neural-networks-3/

Stanford cs231n.

Overview



1. **Forward**: compute output of each layer
2. **Backward**: compute gradient
3. **Update**: update the parameters with computed gradient

Agenda

1. Forward

   Conv, Fully Connected, Pooing, non-linear Function
   Loss functions

2. Back Propagation, Computing Gradient

   Chain rule

3. Updating Parameters

   SGD

4. Training

Batch (batch size)

```
# Training cycle
for epoch in range(training_epochs):
    # Loop over all batches
    for i in range(total_batch):
```

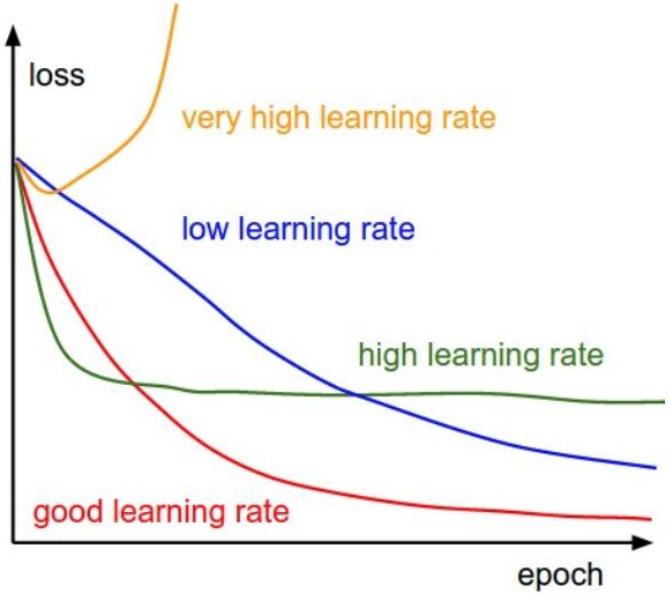
288

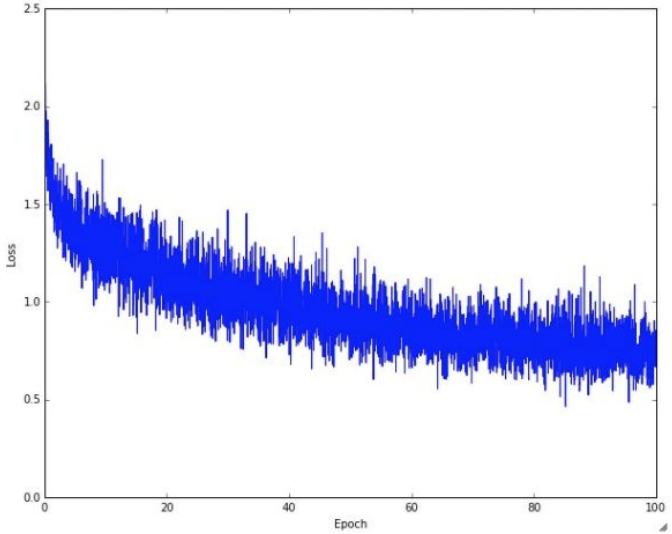
In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

Learning Rate





Left: A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape. Right: An example of a typical loss function over time, while training a small network on CIFAR-10 dataset. This loss function looks reasonable (it might indicate a slightly too small learning rate based on its speed of decay, but it's hard to say), and also indicates that the batch size might be a little too low (since the cost is a little too noisy).

Resources

Resources

Course: Stanford CS231n http://cs231n.stanford.edu/

CS231n: Convolutional Neural Networks for Visual Recognition

Spring 2017

Course Description

Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification, localization and detection. Recent developments in neural network (aka "deep learning") approaches have greatly advanced the performance of these state-of-the-art visual recognition systems. This course is a deep dive into details of the deep learning architectures with a focus on learning end-to-end models for these tasks, particularly image classification. During the 10-week course, students will learn to implement, train and debug their own neural networks and gain a detailed understanding of cutting-edge research in computer vision. The final assignment will involve training a multi-million parameter convolutional neural network and applying it on the largest image classification dataset (ImageNet). We will focus on teaching how to set up the problem of image recognition, the learning algorithms (e.g. backpropagation), practical engineering tricks for training and fine-tuning the networks and guide the students through hands-on assignments and a final course project. Much of the background and materials of this course will be drawn from the ImageNet Challenge.

Resources

Bay Area Deep Learning School https://goo.gl/MvWKYr

Bay Area Deep Learning School Day 1 at CEMEX auditorium, Stanford

102,904 views

1K 14 SHARE

Thank You