
The Ancient
Secrets
Computer Vision

Logistics:

- Homework 4 is out!
  - Experiments out soon on MNIST and CIFAR


Previously
On
the
Ancient Secrets
of Computer Vision

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\begin{aligned}
\partial/\partial v_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial v_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \; \varphi(x_1 w_3 + x_2 w_4)v_2] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -\varphi(x_1 w_3 + x_2 w_4)
\end{aligned}$$

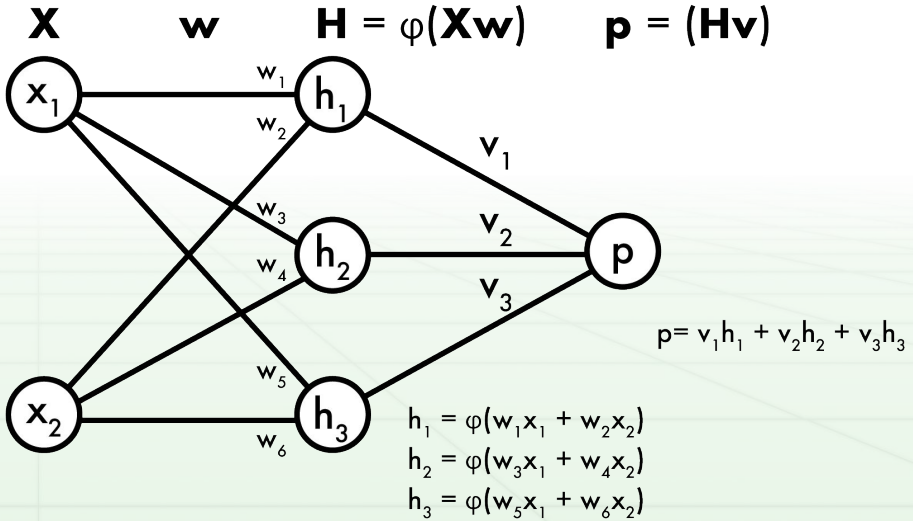
Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\partial/\partial v_2 \; L_{x,Y}(\mathbf{w},\mathbf{v}) = -(Y - \varphi(\mathbf{Xw})\mathbf{v})*h_2$$

Weight update rule (remember descend on loss):

$$v_2 = v_2 + \eta(Y - \varphi(\mathbf{Xw})\mathbf{v})*h_2$$



Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\begin{aligned}
\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1[\partial/\partial w_2 \; \varphi(x_1 w_1 + x_2 w_2)] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1 \varphi'(x_1 w_1 + x_2 w_2)[\partial/\partial w_2 \; (x_1 w_1 + x_2 w_2)] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1 \varphi'(x_1 w_1 + x_2 w_2) * x_2
\end{aligned}$$

Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say w_2:

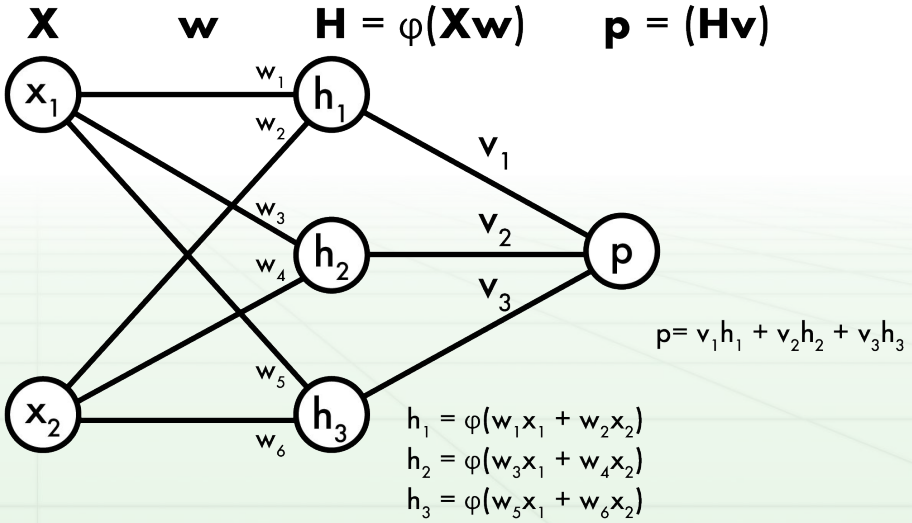
$$\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

Model error at p

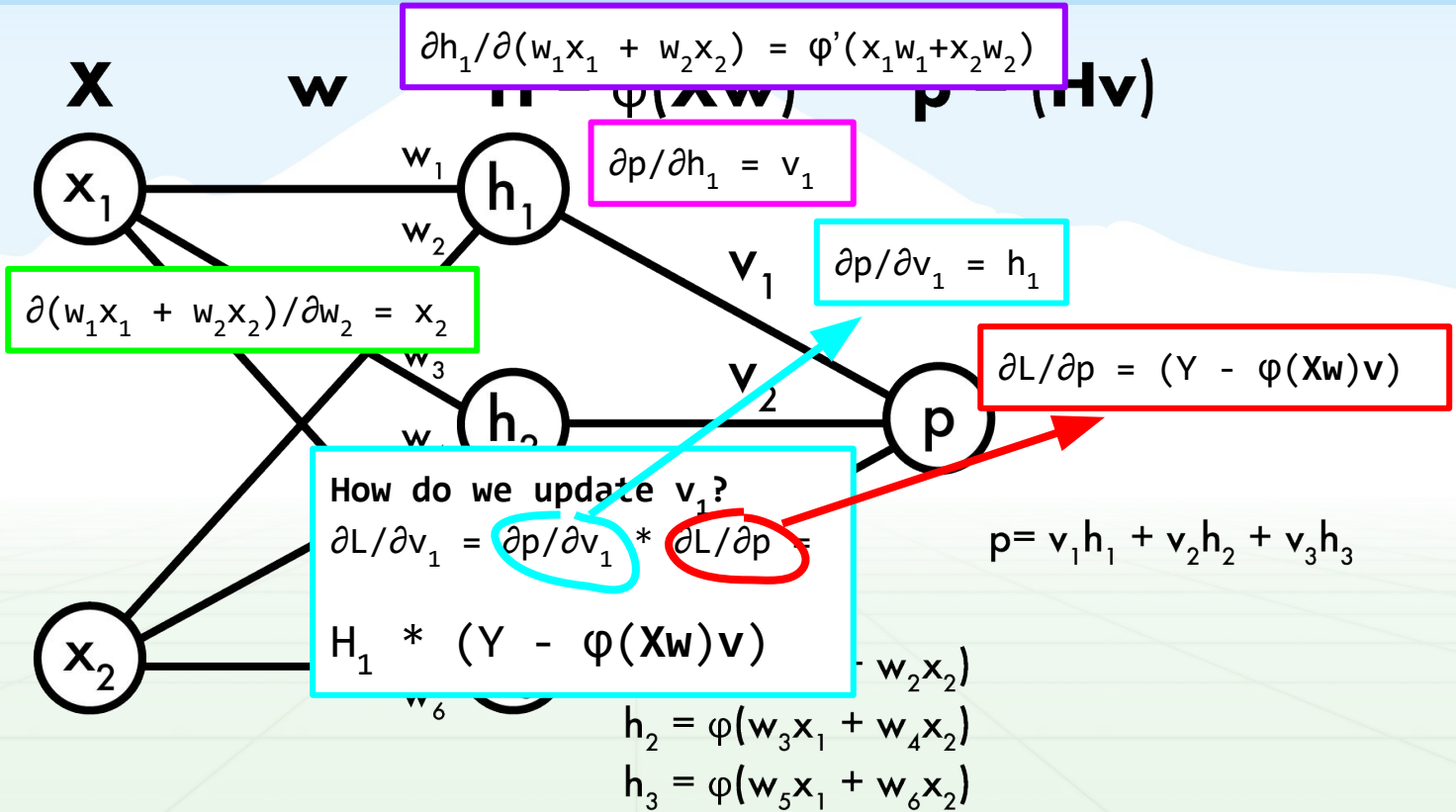
Model error at h_1

Multiply by x_2: gradient w.r.t. w_2

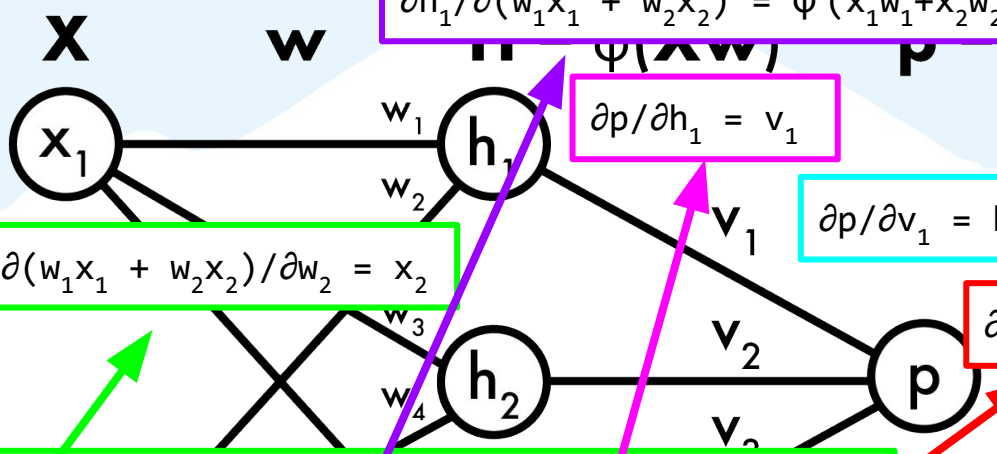
Backpropagate through v_1



Backpropagation: the math



Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{Xw})\mathbf{v})$

$p = v_1h_1 + v_2h_2 + v_3h_3$

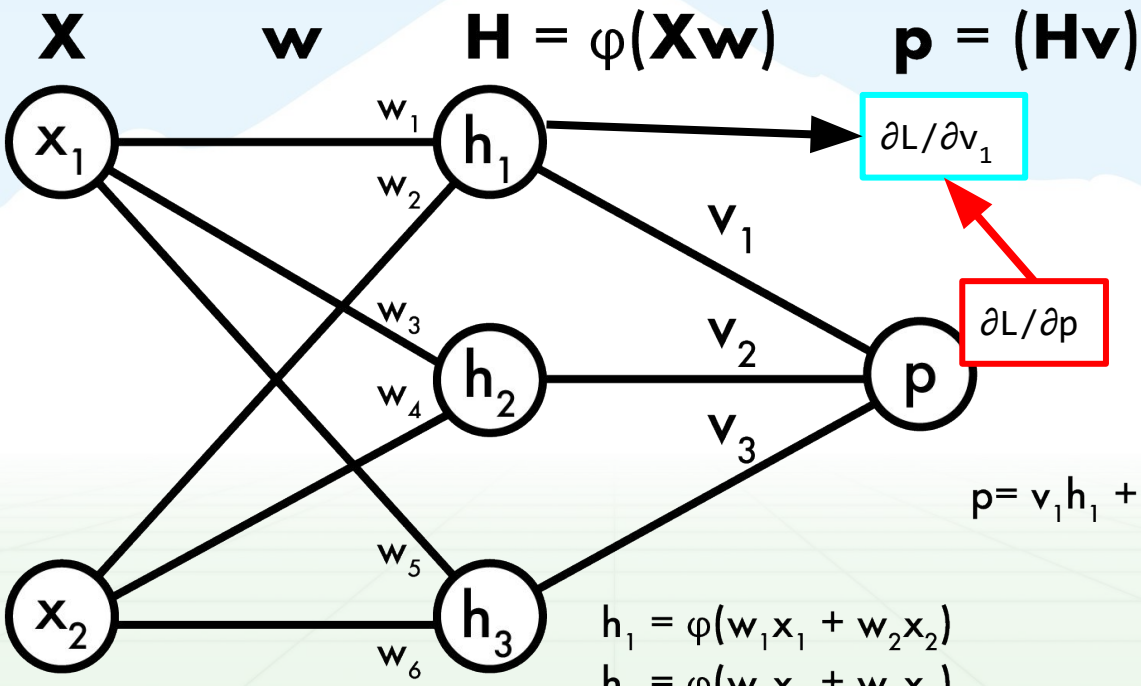
How do we update w_2?

$\partial L/\partial w_2 =$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 * \partial h_1/\partial(w_1x_1 + w_2x_2) * \partial p/\partial h_1 * \partial L/\partial p =$

$x_2 * \varphi'(x_1w_1+x_2w_2) * v_1 * (Y - \varphi(\mathbf{Xw})\mathbf{v})$

Backpropagation: the math



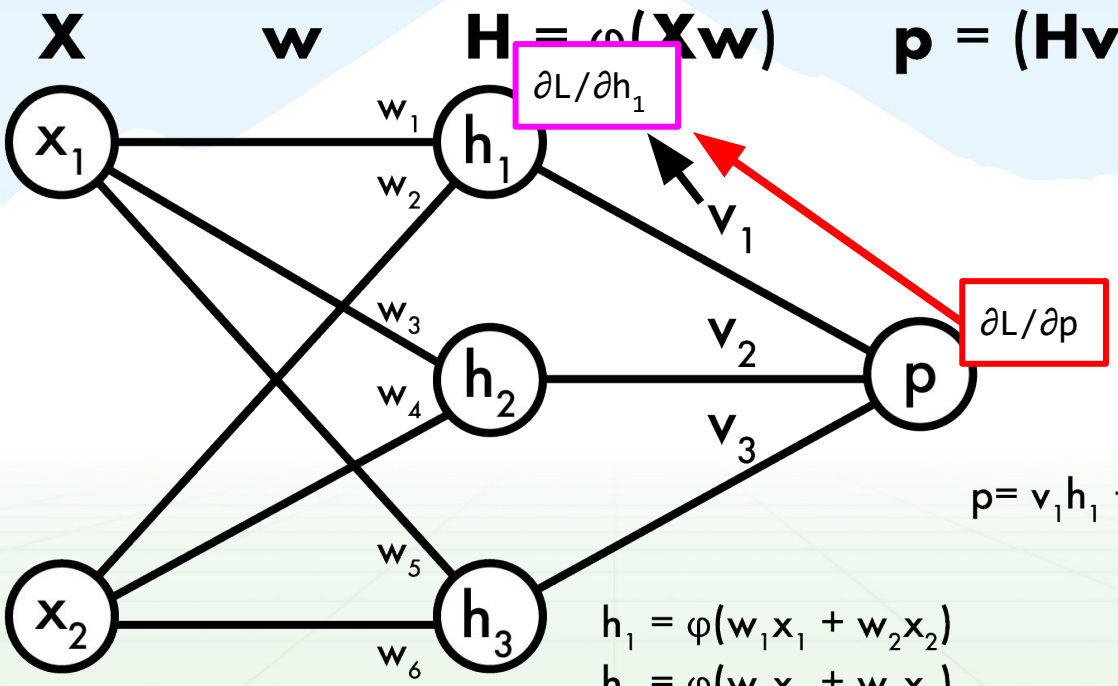
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



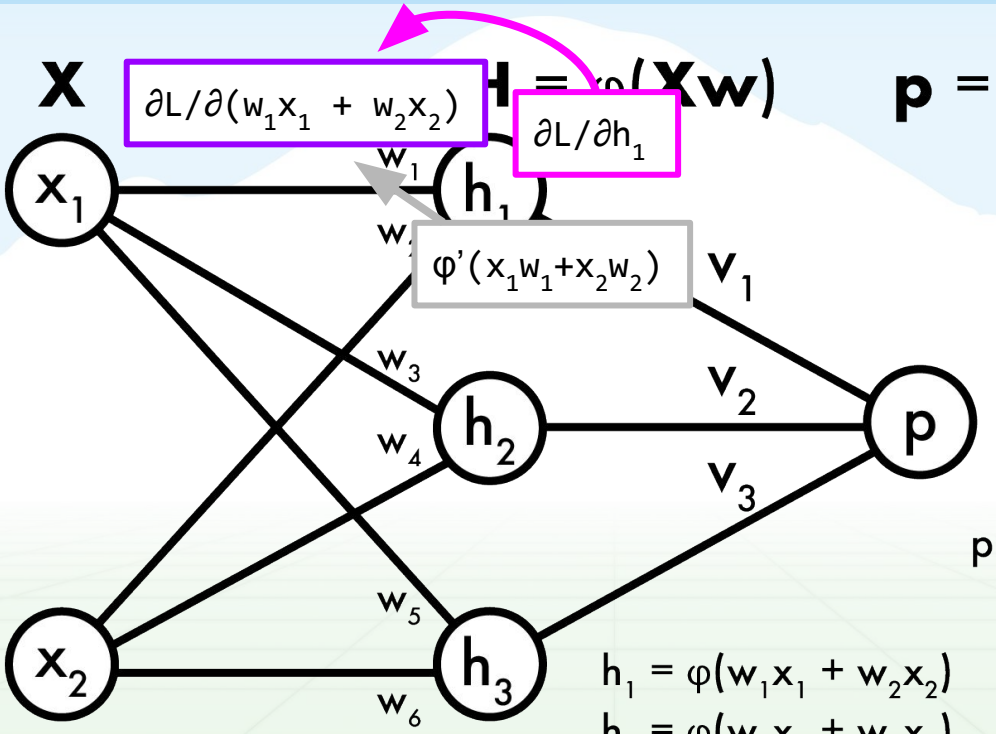
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



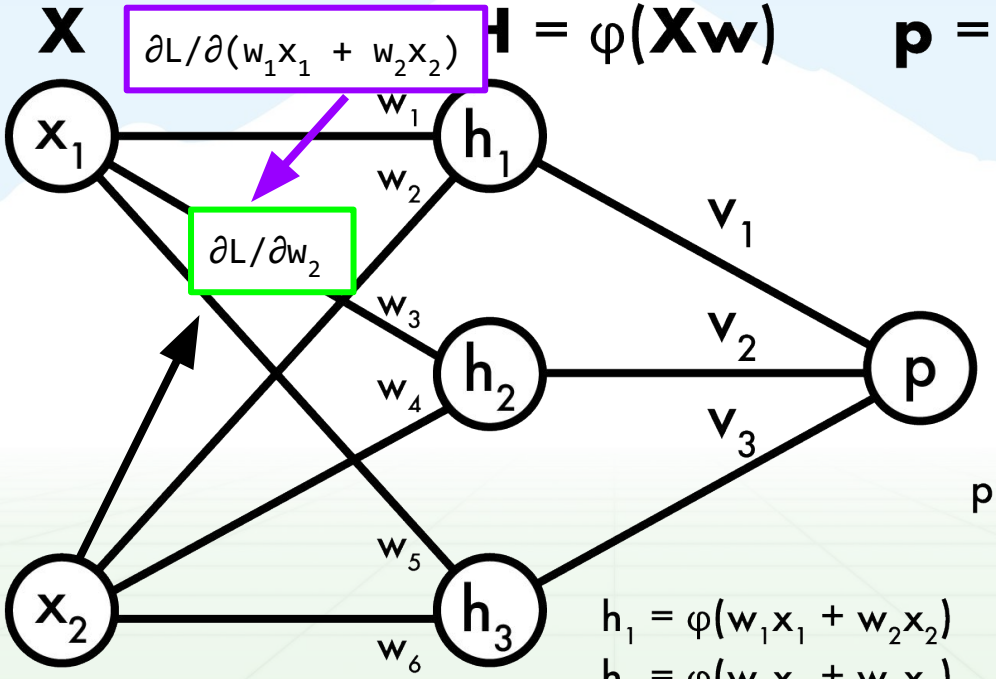
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$p = v_1h_1 + v_2h_2 + v_3h_3$

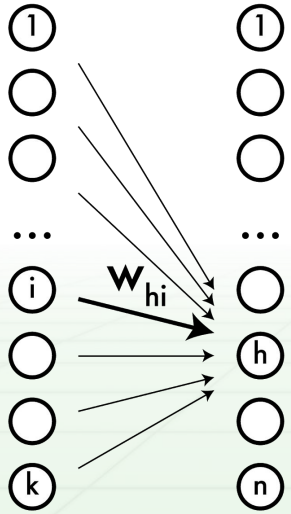
$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

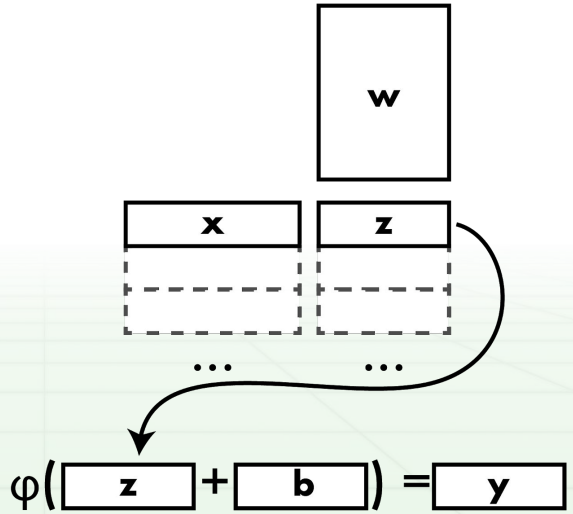
Forward propagation

$$y_h = \varphi(\sum x_i w_{hi} + b)$$



Layer x Layer y

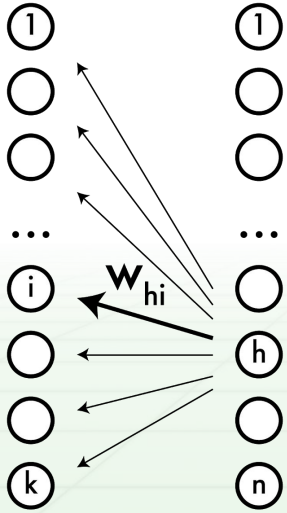
$$\mathbf{y} = \varphi(\mathbf{xw} + \mathbf{b})$$

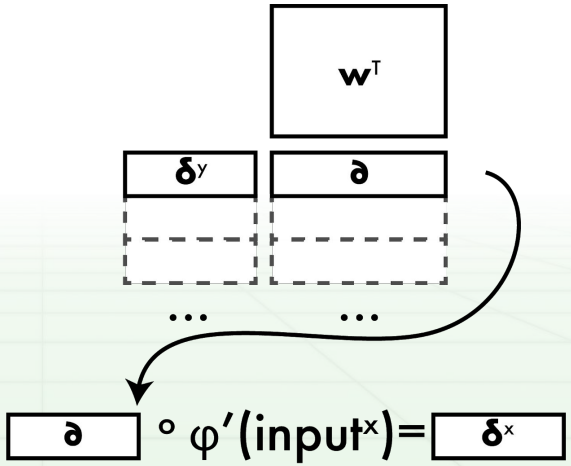


x: m×k **w**: k×n **z**: m×n

Backward propagation

$$\delta_i^x = \left(\sum \delta_j^y w_{hi}\right) \varphi'(input_i^x) \qquad \boldsymbol{\delta}^x = \boldsymbol{\delta}^y \mathbf{w}^T \circ \varphi'(input^x)$$



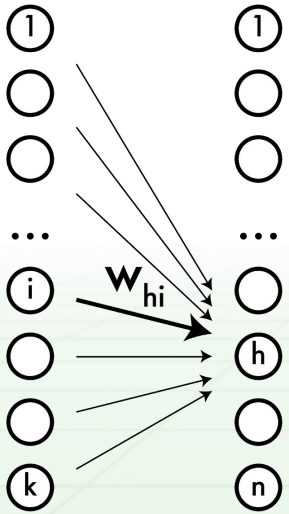


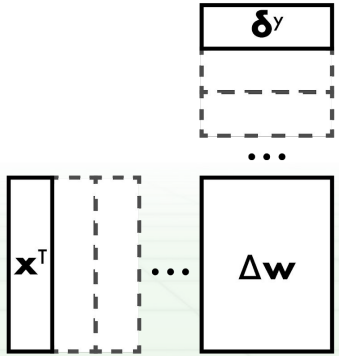
Layer x Layer y $\boldsymbol{\delta}^y$: m×n $\mathbf{w}^T$: n×k $\boldsymbol{\delta}^x$: m×k

Weight updates

$$\Delta w_{hi} = \delta_h^y x_i$$

$$\Delta \mathbf{w} = \mathbf{x}^T \boldsymbol{\delta}^y$$





Layer x Layer y $\mathbf{x}^T$: k×m $\boldsymbol{\delta}^y$: m×n $\Delta \mathbf{w}$: k×n

Weight decay: neural network regularization

$\text{argmin}_w L_x(\mathbf{w}) + \lambda ||\mathbf{w}||_2$

λ: regularization parameter

Higher: more penalty for large weights, less powerful model

Lower: less penalty, more overfitting

Commonly use L_2 norm to regularize, *weight decay*

Gradient descent update rule:

$$w_{t+1} = w_t - \eta[\partial/\partial w_t \, L(w_t) + \lambda w_t]$$

$$= w_t - \eta\partial/\partial w_t \, L(w_t) - \eta\lambda w_t$$

Subtract a little bit of weight every iteration

Momentum: speeding up SGD

If we keep moving in same direction we should move further every round

Before:

$$\Delta w_t = -\partial/\partial w_t \; L(w_t)$$

Now:

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) + m\Delta w_{t-1}$$

$$w_{t+1} = w_t + \eta \Delta w_t$$

Side effect: **smooths** out updates if gradient is in different directions

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

Gradient of loss

Weight decay

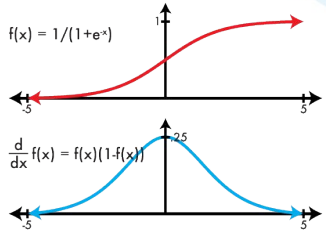
Momentum

$$w_{t+1} = w_t + \eta \Delta w_t$$

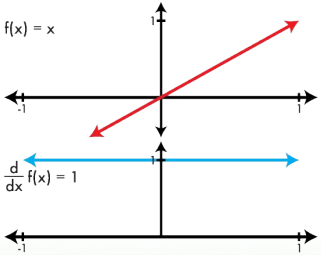
Learning rate

Common activation functions φ

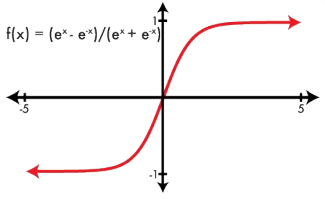
logistic



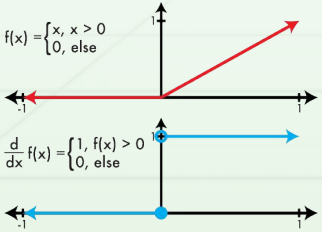
linear



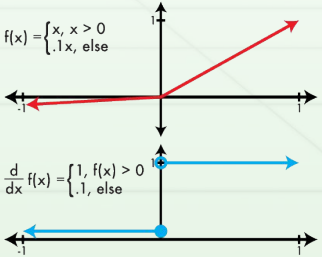
tanh



REctified Linear Unit (RELU)



Leaky RELU



Hyper Parameter Dark Magic

What follows are the one, true, correct, and only set of hyperparameters. Praise be the NetLord!

$\eta = [.0001 - .01]$

$\lambda = .0005$ <- at least for vision

$m = .9$

$\varphi =$ leaky relu


Chapter
Thirteen
Convolutional
Neural Networks

Neural networks and images

Neural networks are densely connected

- Each neuron in layer i connected to every neuron in layer i+1

HOG features: 36 features / 8 x 8 patch

Say we want to process images:

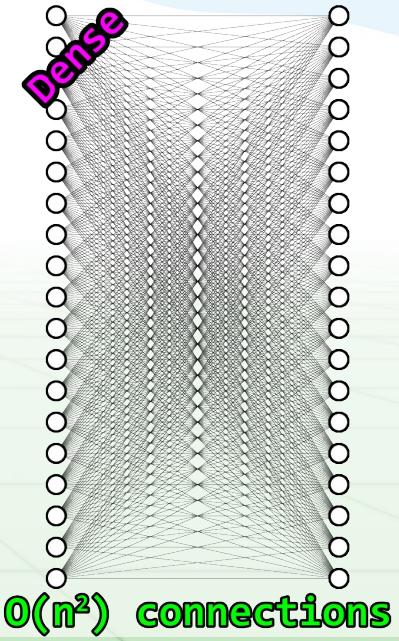
- Input : 256 x 256 x 3 RGB image
- Hidden : 32 x 32 x 36 feature map?
- Output : 1000 classes

Input -> hidden is 7.2 billion connections!

Too many weights!

Neural networks are densely connected

But is this really what we want when processing images?

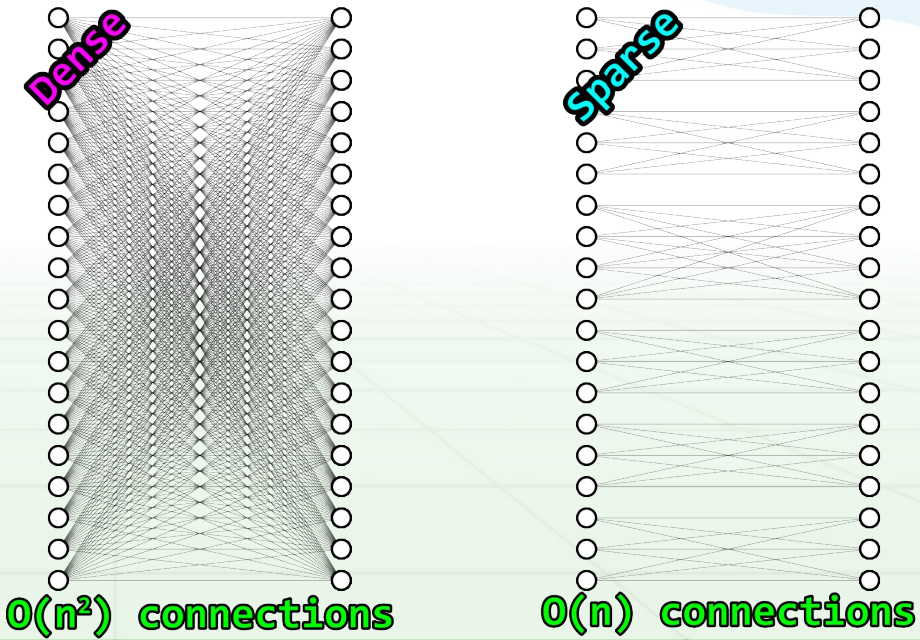


Too many weights!

Would rather have sparse connections

- Fewer weights
- Nearby regions - related
- Far apart - not related

How can we do this?



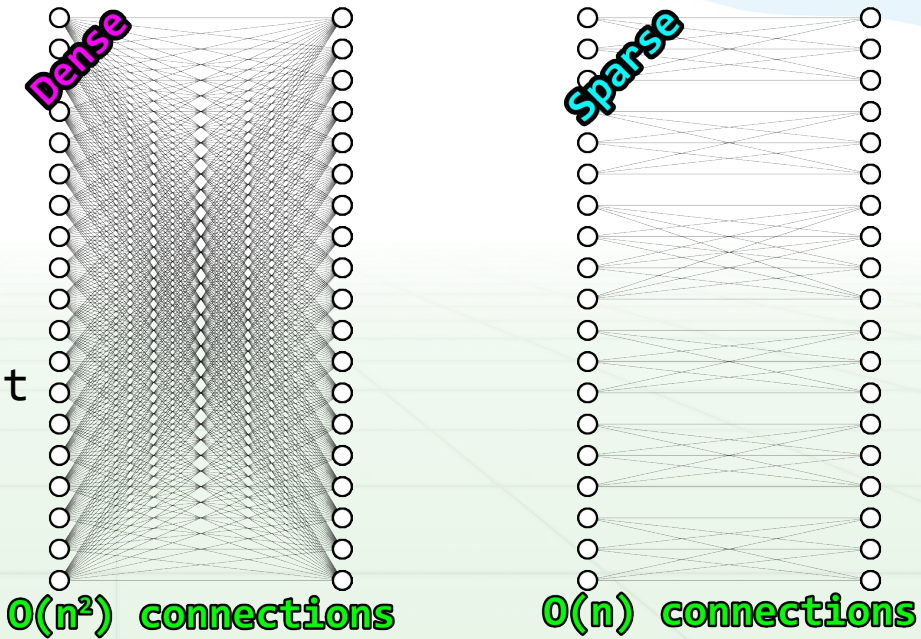
Too many weights!

Would rather have sparse connections

- Fewer weights
- Nearby regions - related
- Far apart - not related

Convolutions!

- Just weighted sums of small areas in image
- Weight sharing in different locations in image



Convolutional neural networks

Use convolutions instead of dense connections to process images

Takes advantage of structure in our data!

Imposes an assumption on our model:

Nearby pixels are related, far apart ones are less related.

What does this do to our bias/variance??

Convolutional Layer

Input: an image

Processing: convolution with multiple filters

Output: an image, # channels = # filters

Output still weighted sum of input (w/ activation)

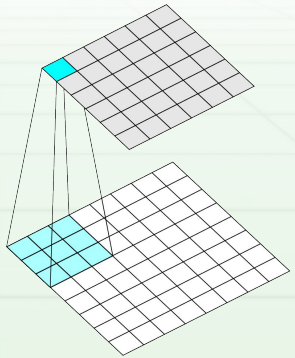
Kernel size

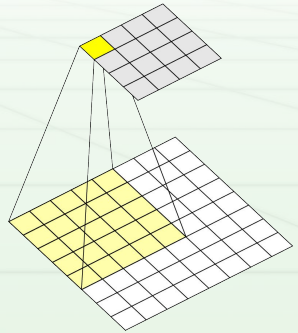
How big the filter for a layer is

Typically 1 x 1 <-> 11 x 11

1 x 1 is just linear combination of channels in previous image (no spatial processing)

Filters usually have same number of channels as input image.



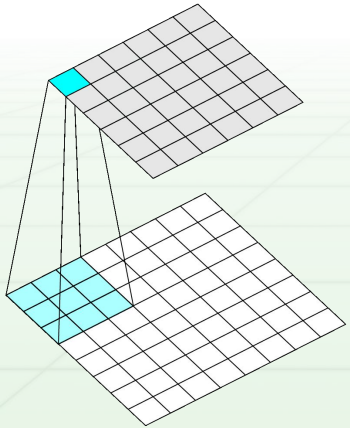


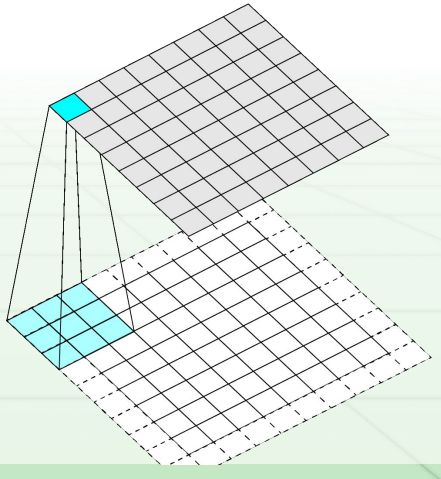
Padding

Convolutions have problems on edges

Do nothing: output a little smaller than input

Pad: add extra pixels on edge





Stride

How far to move filter between applications

We’ve done stride 1 convolutions up until now, approximately preserves image size

Could move filter further, downsample image

Implement using matrices!

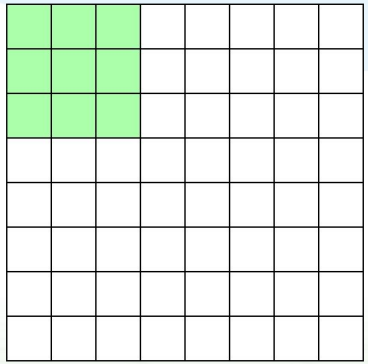
We want convolution to be matrix operations because matrices are fast. How?

Im2col: rearrange image

Take spatial blocks of image and put them into columns of a matrix.

Im2col handles kernel size, stride, padding, etc.

Makes matrix with all relevant pixels in the image.



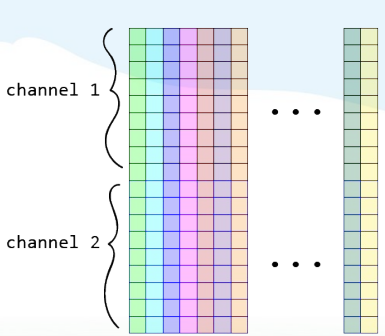
Im2col: rearrange image

Take spatial blocks of image and put them into columns of a matrix.

Im2col handles kernel size, stride, padding, etc.

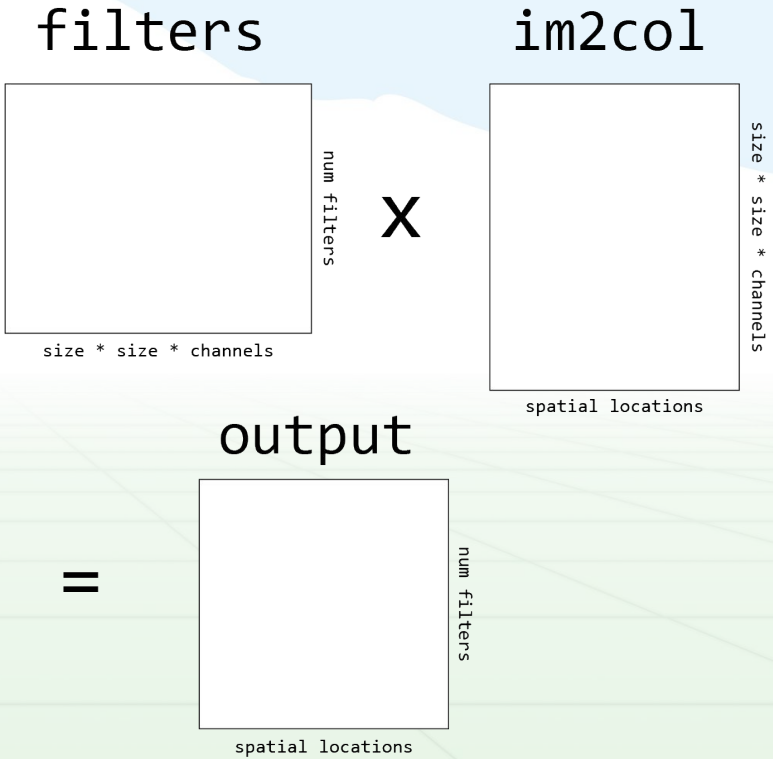
Makes matrix with all relevant pixels in the image.

Multiple channel image stacked by channel.



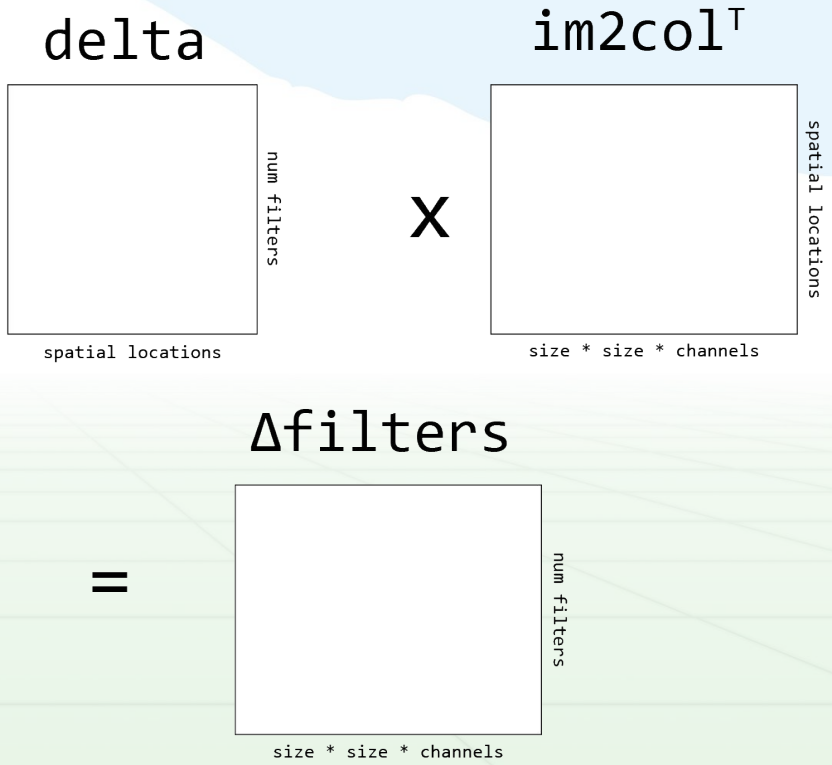
Im2col: rearrange image

Now we just multiply by our filter matrix to do convolutions.



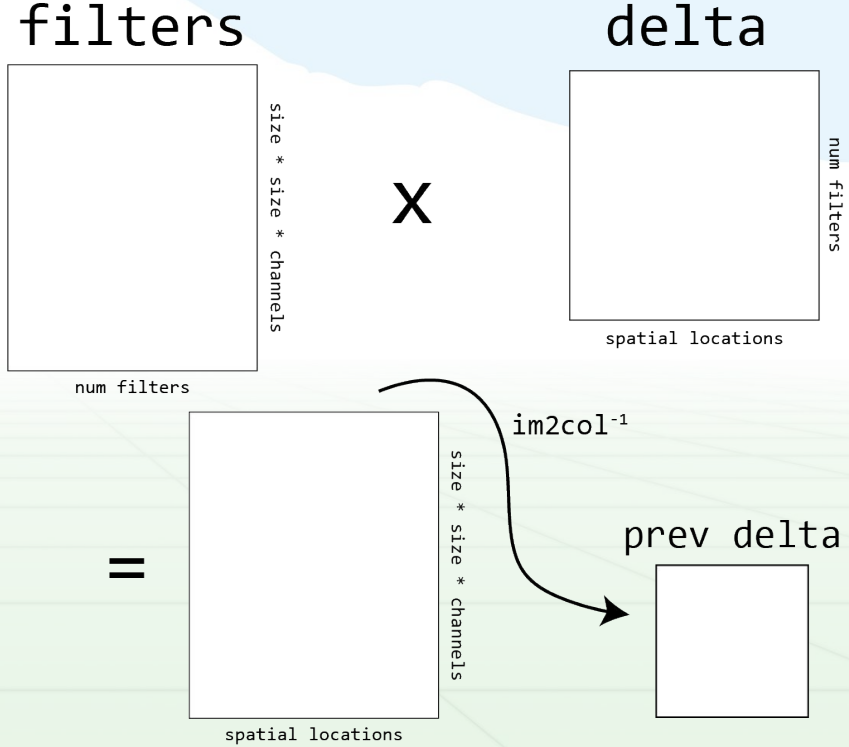
Im2col: rearrange image

Can calculate our weight updates as well by multiplying the delta of a layer by the input.



Im2col: rearrange image

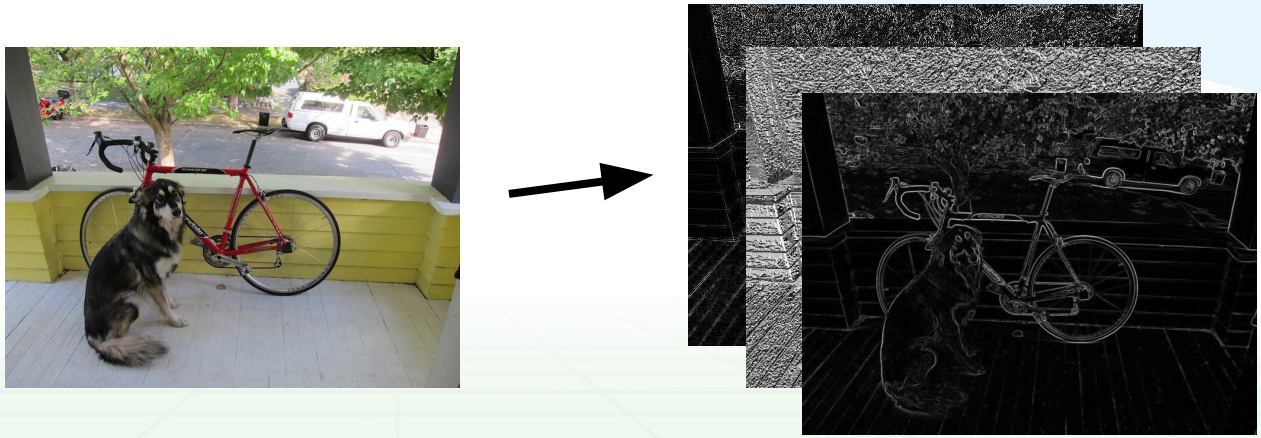
Finally, can calculate backpropagated delta by multiplying the current delta by weights and doing the reverse of im2col.



Images are BIG

Even a 256 x 256 images has hundreds of thousands of pixels and that’s considered a small image!

Convolution:



Aggregate information, maybe we don’t need all of the image, can subsample without throwing away useful information

Pooling Layer

Input: an image

Processing: pool pixel values over region

Output: an image, shrunk by a factor of the stride

Hyperparameters:

- What kind of pooling? Average, mean, max, min
- How big of stride? Controls downsampling
- How big of region? Usually not much bigger than stride

Most common: 2x2 or 3x3 maxpooling, stride of 2

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7		

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7		

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8		

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9		

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10
8			

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10
8	7		

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10
8	7	4	

Maxpooling Layer, 2x2 stride 2

-7	6	-1	3	9	9	6	-9
3	-8	0	7	10	8	-3	10
-4	2	-6	4	-7	5	5	7
-3	-9	1	8	-8	9	-1	-5
-7	10	-9	-5	9	-8	-7	10
-5	5	9	4	10	-8	7	6
-3	8	0	2	2	-3	-2	5
4	-6	7	-3	1	4	10	0

6	7	10	10
2	8	9	7
10	9	10	10
8	7	4	10

(Fully) Connected Layer

The standard neural network layer where every input neuron connects to every output neuron

Often used to go from image feature map -> final output or map image features to a single vector

Eliminates spatial information

Convnet Building Blocks

Convolutional layers:

- Connections are convolutions
- Used to extract features

Pooling layers:

- Used to downsample feature maps, make processing more efficient
- Most common: maxpool, avgpool sometimes used at end

Connected layers:

- Often used as last layer, to map image features -> prediction
- No spatial information
- Inefficient: lots of weights, no weight sharing