CNN Architectures

Computer Vision

Winter Semester 20/21

Goethe University

Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, etc.

What we did last week

- Deep Learning
- Convolutional Neural Networks

Today's class

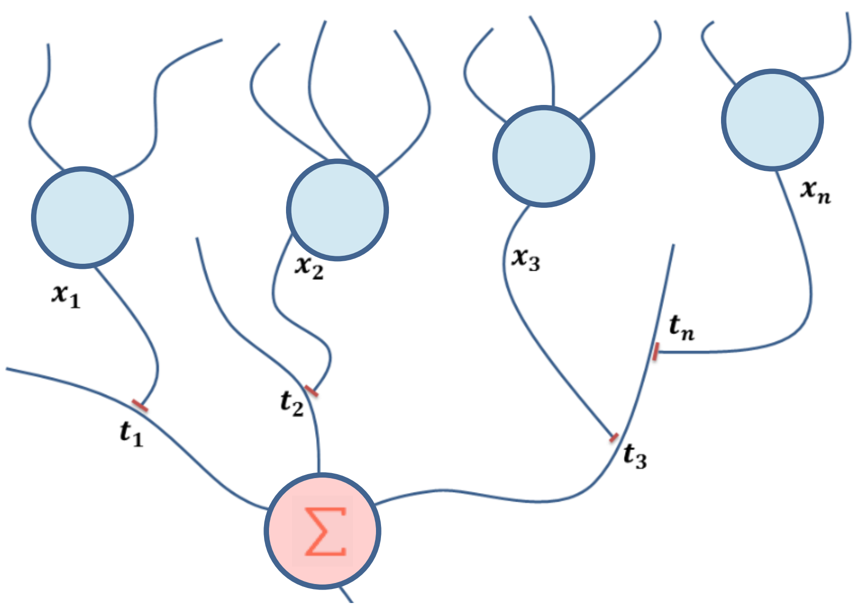
- CNN architectures:
    - LeNet
    - AlexNet
    - GoogleLeNet (Inception)
    - ResNet

- Backpropagation

CNN – inspired by neuroscience

Simplified neuroscience: a neuron computes a dot product between its inputs and the synaptic weights

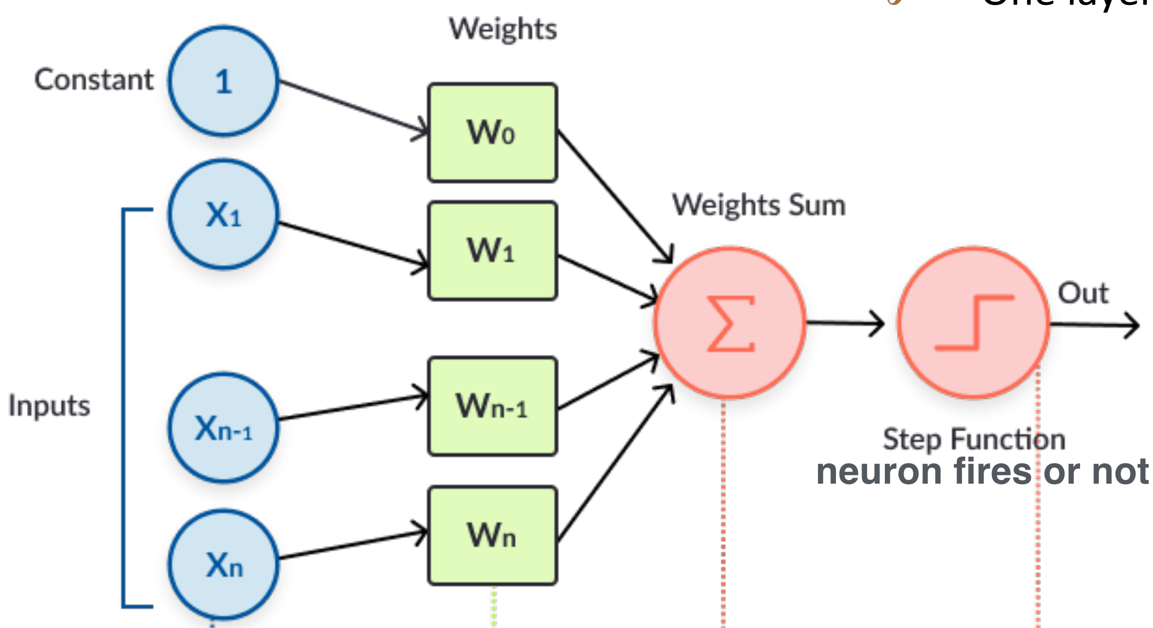
$$< x, t > = \sum_{i=1}^{n} x_i t_i \longleftrightarrow$$



Simple perceptron

F. Rosenblatt 1957

- One layer NN



Input +
constant for bias

Weights
Learned

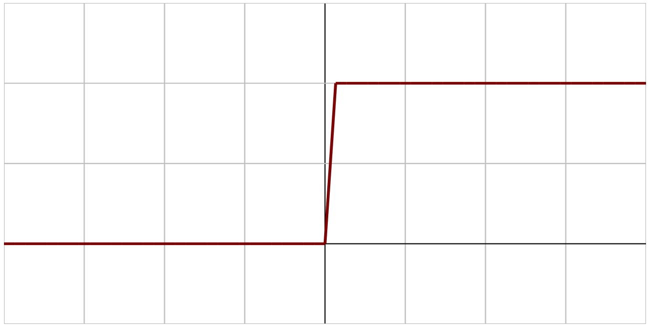
$$\sum_{i=0}^{n} x_i w_i$$

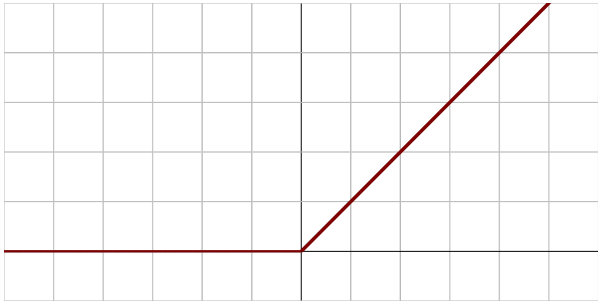
$$Out = sgn\left(\sum_{i=0}^{n} x_i w_i\right)$$

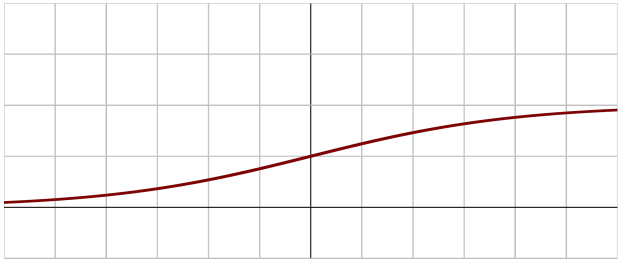
Image credit: missinglink.ai

Simple perceptron

Types of Nonlinearities







Step function Linear Rectifier (ReLu) Sigmoid

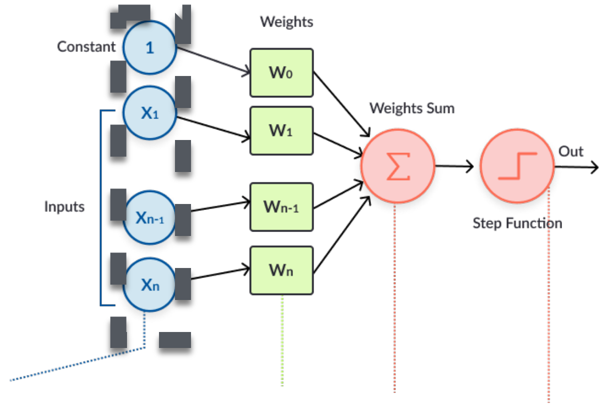
$$f(x) = \begin{cases} 0 & : x < 0 \\ 1 & : x \geq 0 \end{cases}$$

$$f(x) = \begin{cases} 0 & : x < 0 \\ x & : x \geq 0 \end{cases}$$

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

etc.

Simple perceptron

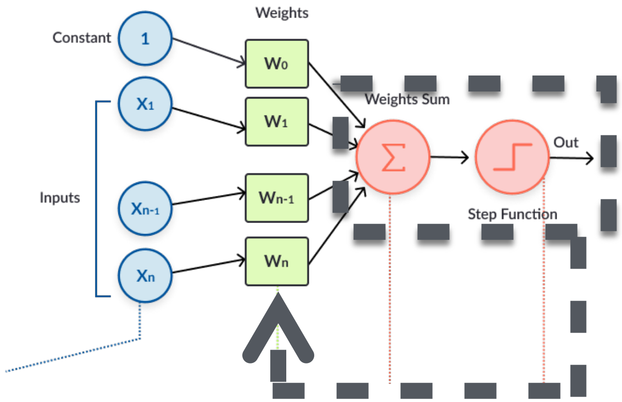


Given training samples $\{\mathbf{x}_i, y_i\}_{\forall i}$

$\mathbf{x}_i$ -> input of example *i*,

y_i -> groundtruth target of example *i*

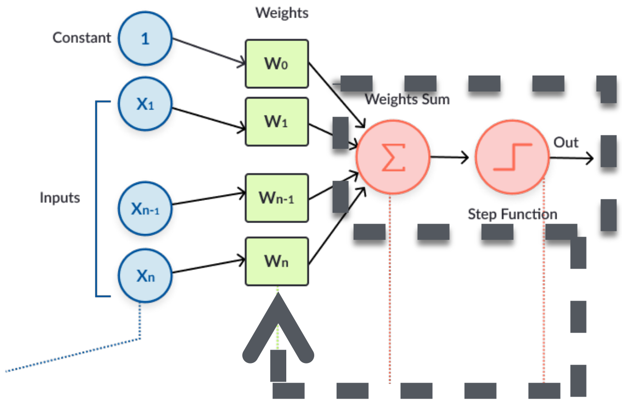
Simple perceptron



Initialization:

Initialize the weights W to 0 or small random numbers.

Simple perceptron



Initialization:

Initialize the weights W to 0 or small random numbers.

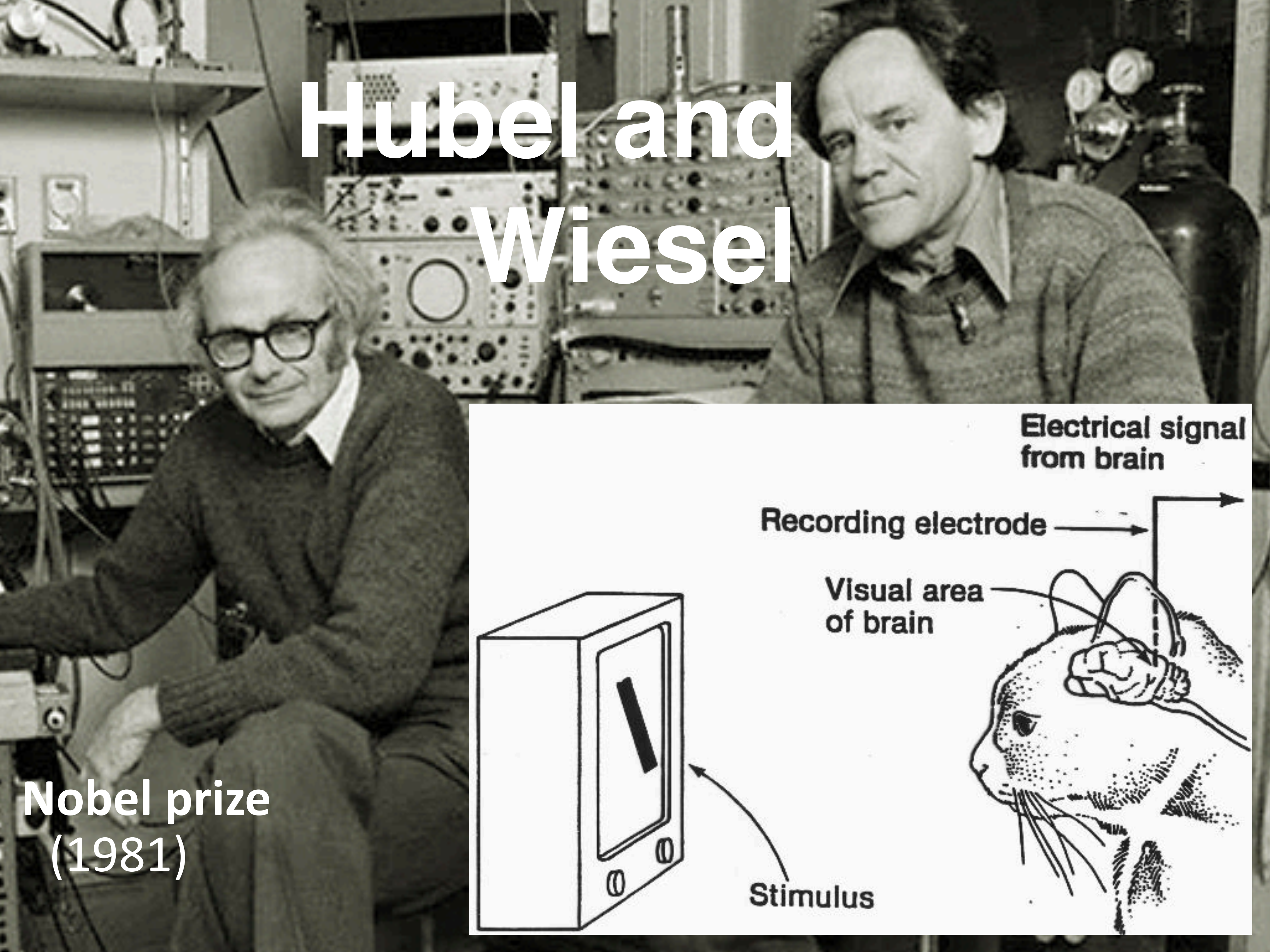
Iterate:

For each training sample $\mathbf{x}_i$:

1. Calculate the output value: $out = sgn(\sum_{i=0}^{n} x_i w_i)$
2. Update the weights. $\mathbf{w} = \mathbf{w} + \eta \mathbf{x_i}(y_i - out)$

In case of linear separable data, the learning converges in a bounded number of iterations.

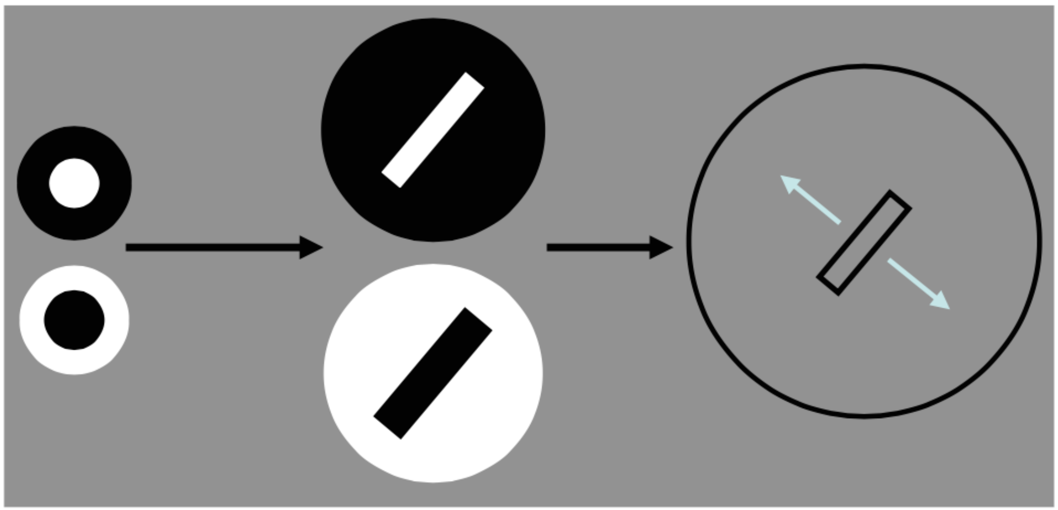
Hubel and Wiesel

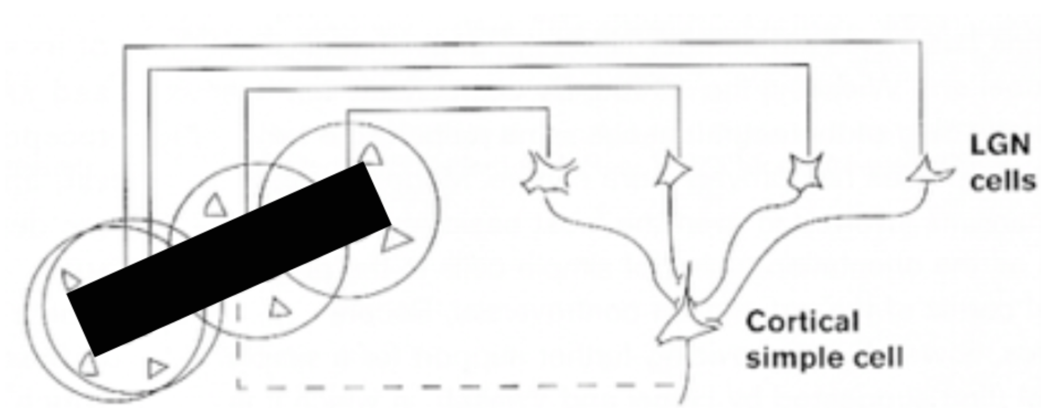


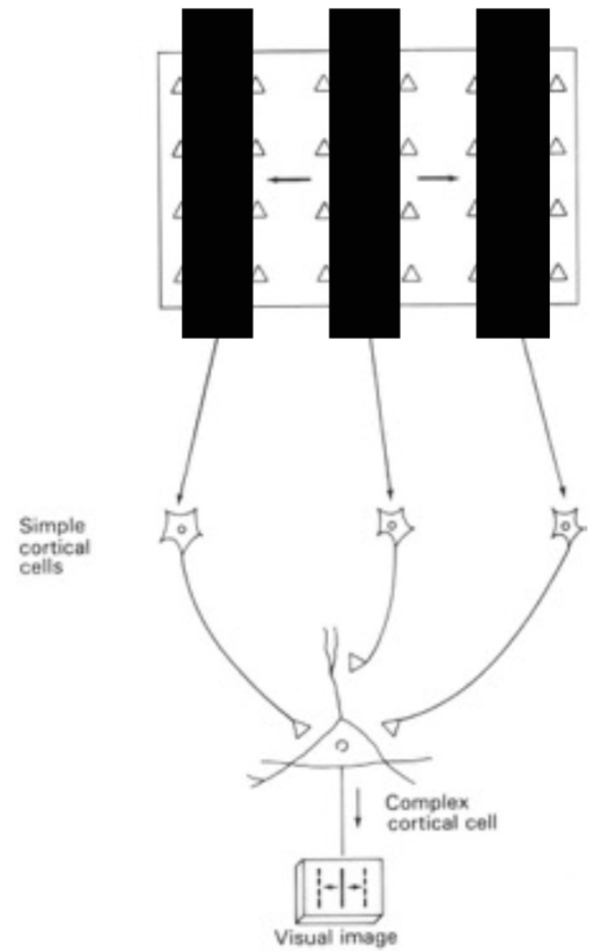
Nobel prize
(1981)

Hubel and Wiesel

LGN–type cells　Simple cells　Complex cells







(Hubel & Wiesel 1959)

Simple and Complex Cells

- Tuning operation (Gaussian-like, AND-like)

$$y = e^{-|x-w|^2}$$

$$or$$

$$y \sim \frac{x \bullet w}{|x|}$$

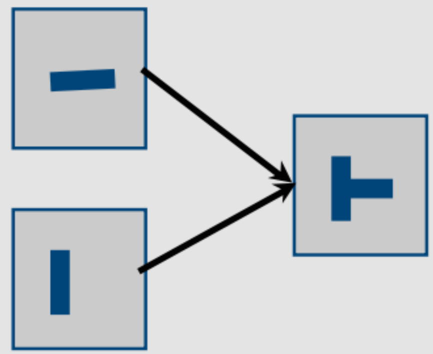
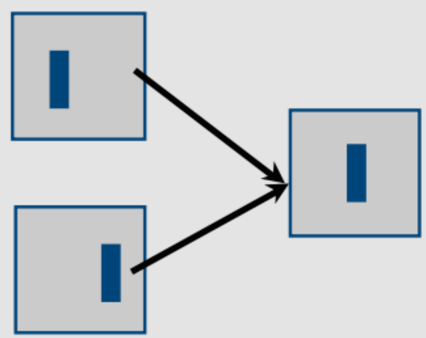
- Simple units

- Max-like operation (OR-like)

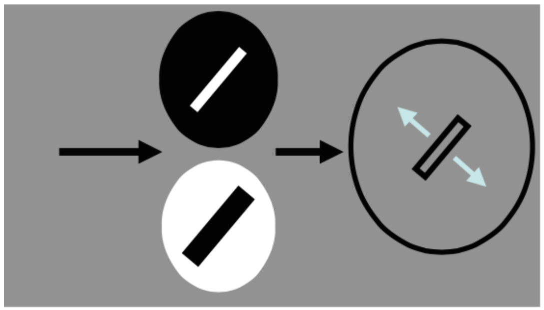
$$y = \max\{x1, x2, \ldots\}$$

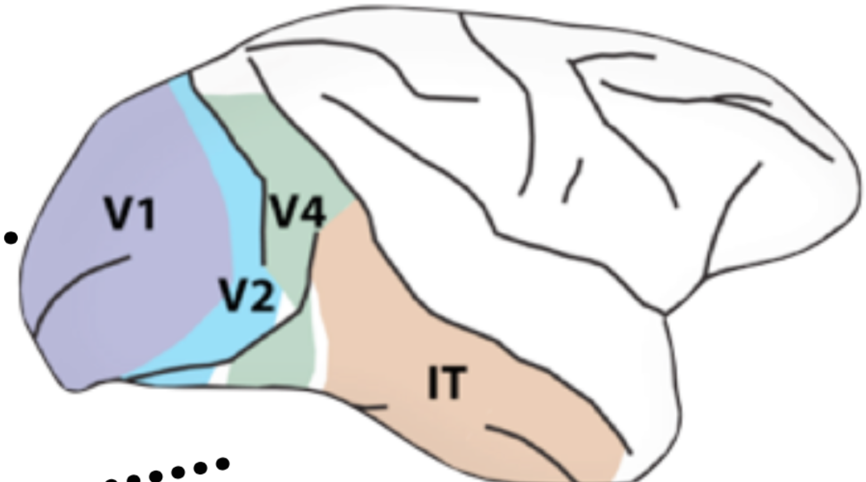
- Complex units

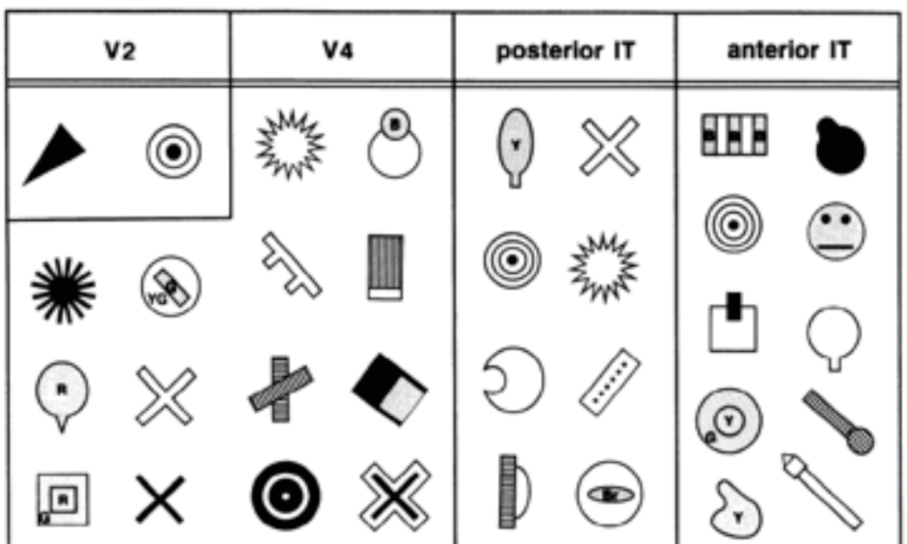
Simple and Complex Cells

Unit	Pooling	Computation
Simple		Selectivity / template matching
Complex		Invariance

The visual ventral stream



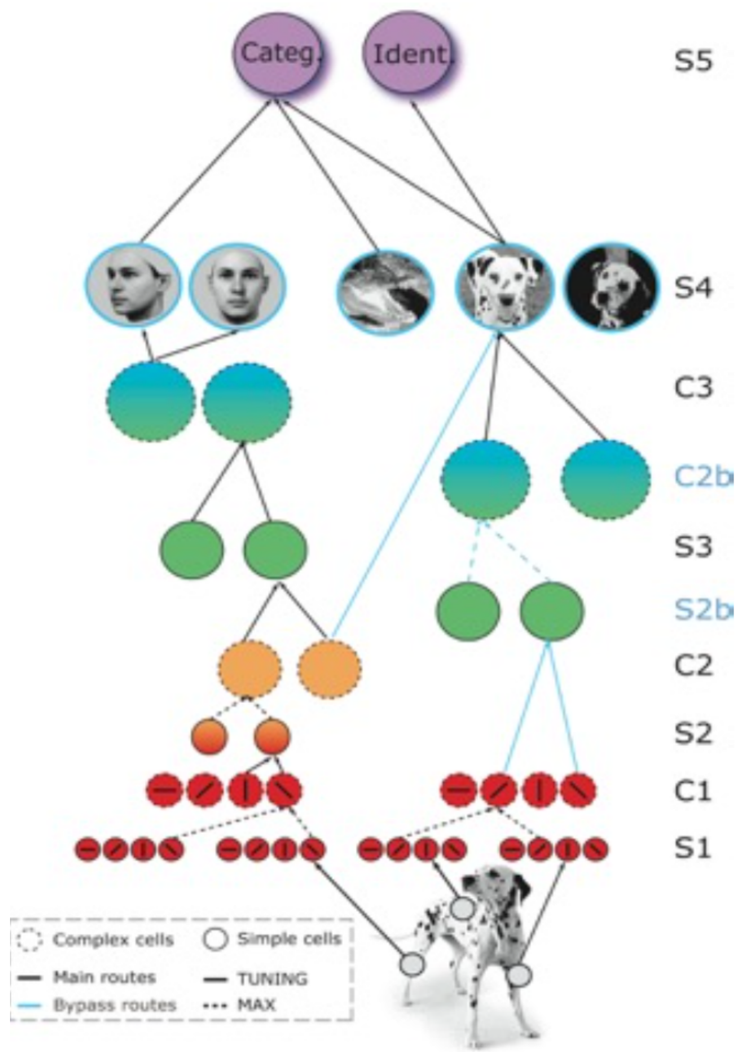




The ventral stream hierarchy: V1, V2, V4, IT
A gradual increase in the receptive field size, in the complexity of the preferred stimulus, in tolerance to position and scale changes

Kobatake & Tanaka, 1994

HMAX



Riesenhuber & Poggio 1999, 2000; Serre Kouh Cadieu Knoblich Kreiman & Poggio 2005; Serre Oliva Poggio 2007

Two operations (~OR, ~AND): disjunctions of conjunctions

- Tuning operation (Gaussian-like, AND-like) $y = e^{-|x-w|^2}$

  or

  $$y \sim \frac{x \bullet w}{|x|}$$

- Simple units

- Max-like operation (OR-like)

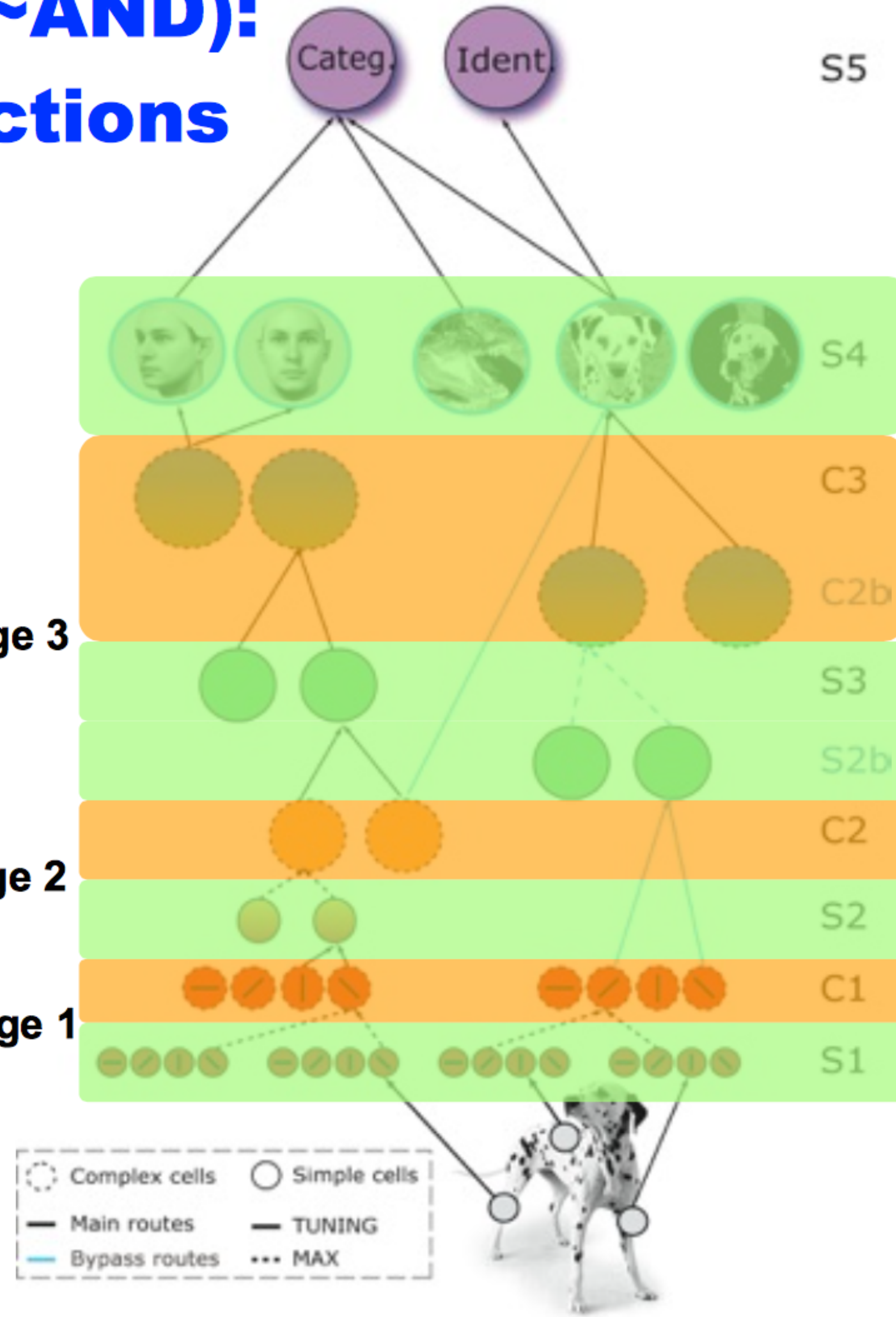
  $$y = \max\{x1, x2, \ldots\}$$

- Complex units

Stage 3

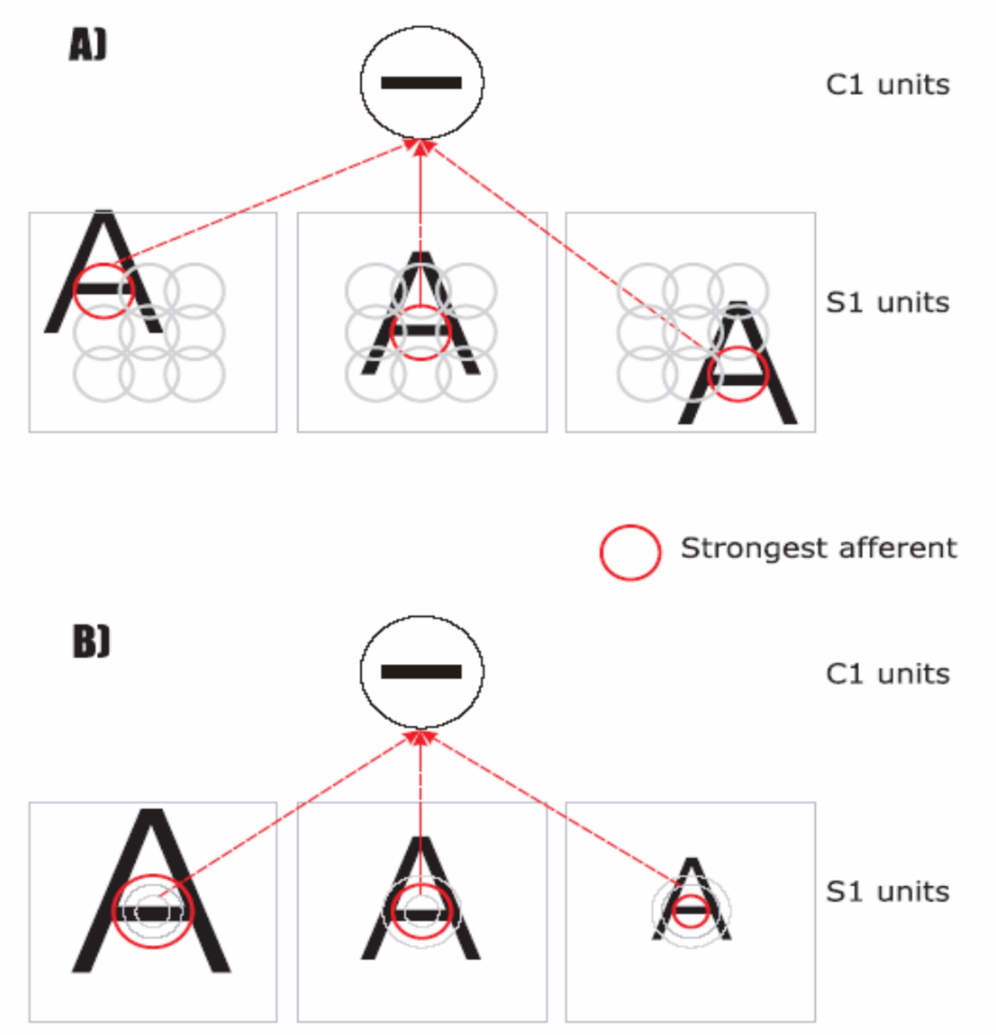
Stage 2

Stage 1



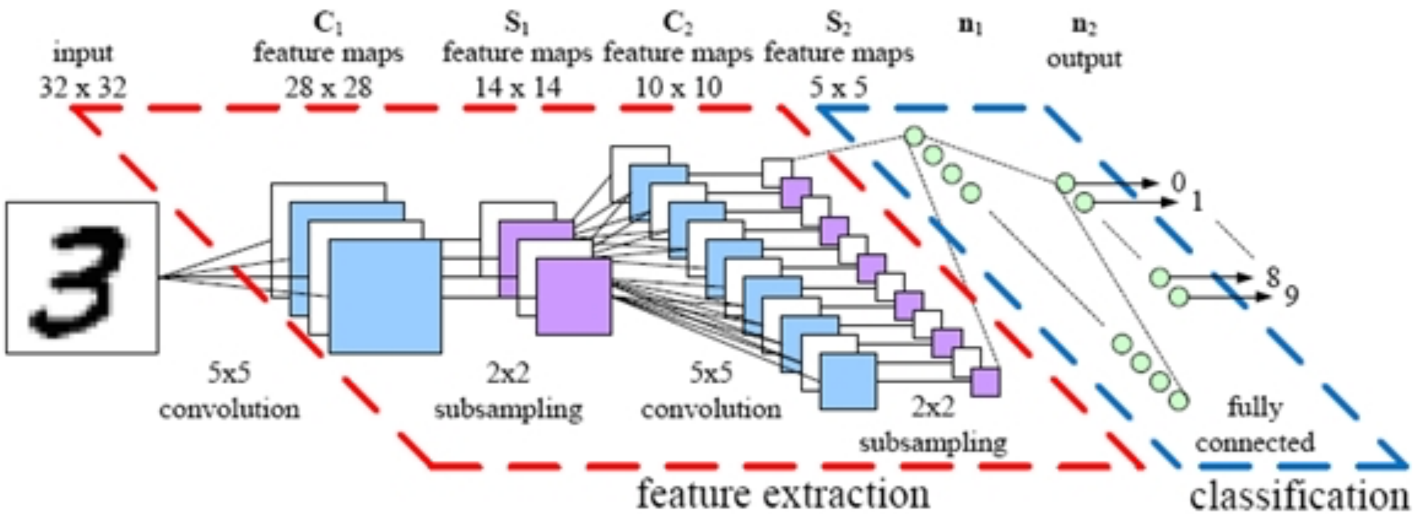
Each operation ~microcircuits of ~100 neurons

Invariance



Serre, T., and Riesenhuber, M. (2004)

Convolutional Neural Networks (CNNs)

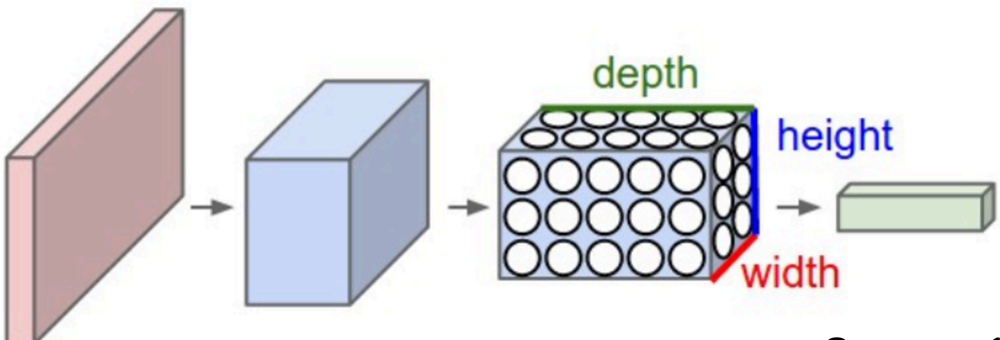


Convolutional assumption

LeCun et al. 98

CNN

- 3D volumes of neurons



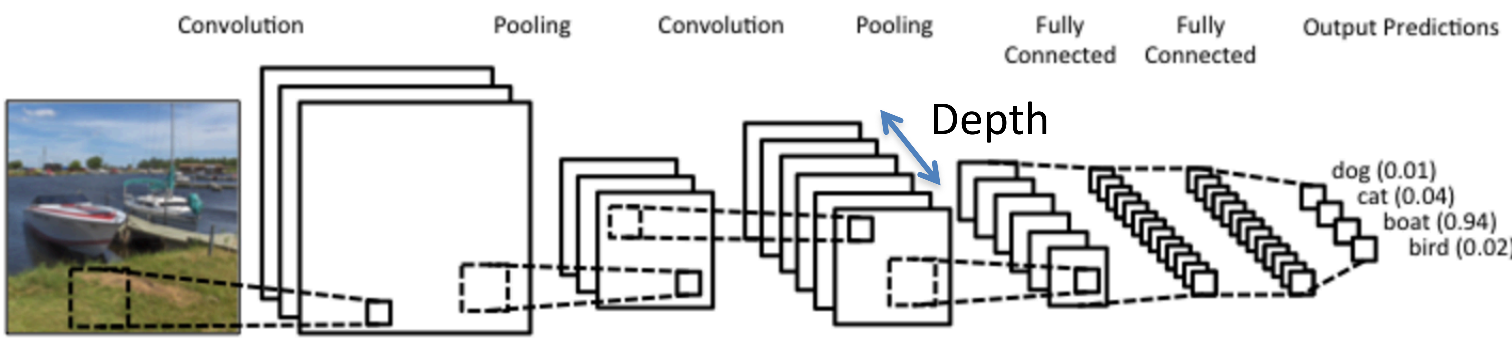
Input: width x height x numChannel

Output: 1 x 1 x numClass

- Stack of
  - Convolutional layer
  - Fully connected layer
  - Pooling layer

CNN

- One example



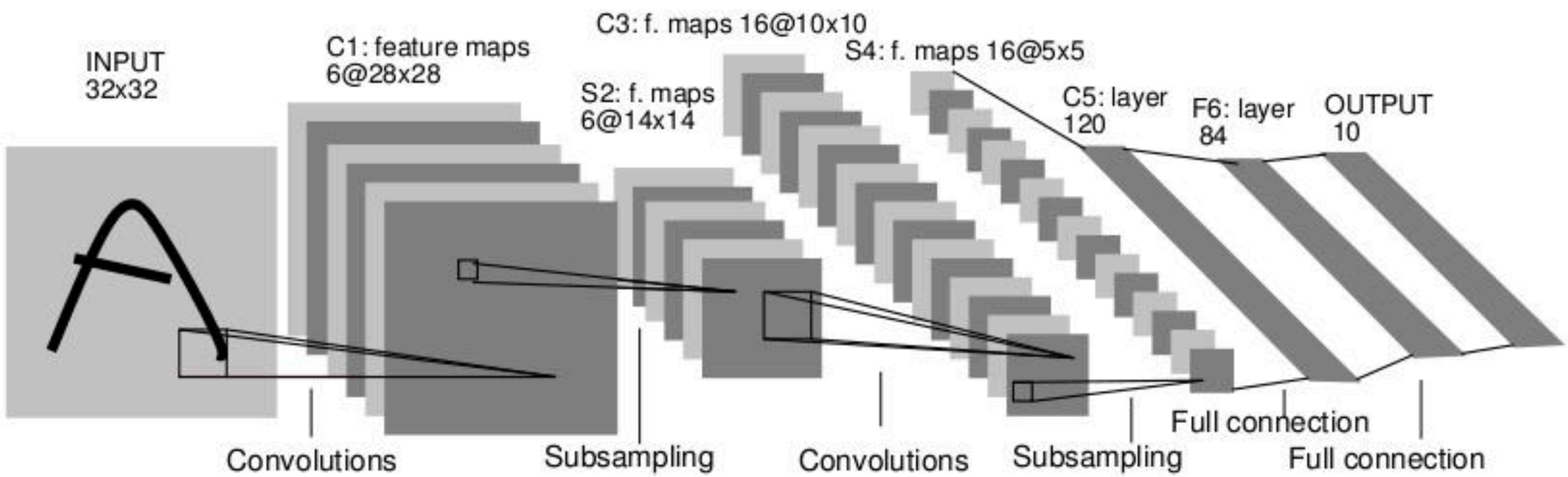
Input:
width x height x numChannel

Output:
1 x 1 x numClass

- Stack of
  – Convolutional layer
  – Fully connected layer
  – Pooling layer

CNN Architecture

- LeNet for character recognition

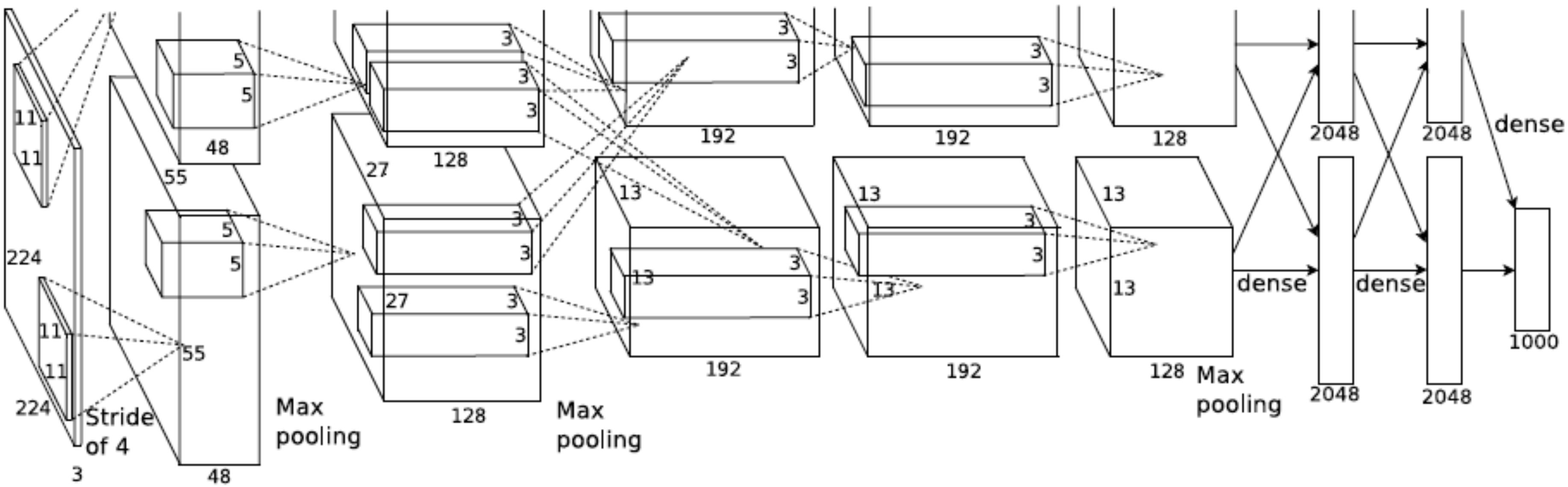


-Average pooling
-Sigmoid or tanh nonlinearity
-Trained on MNIST digit dataset (60K training examples)

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner,
Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998

CNN Architecture

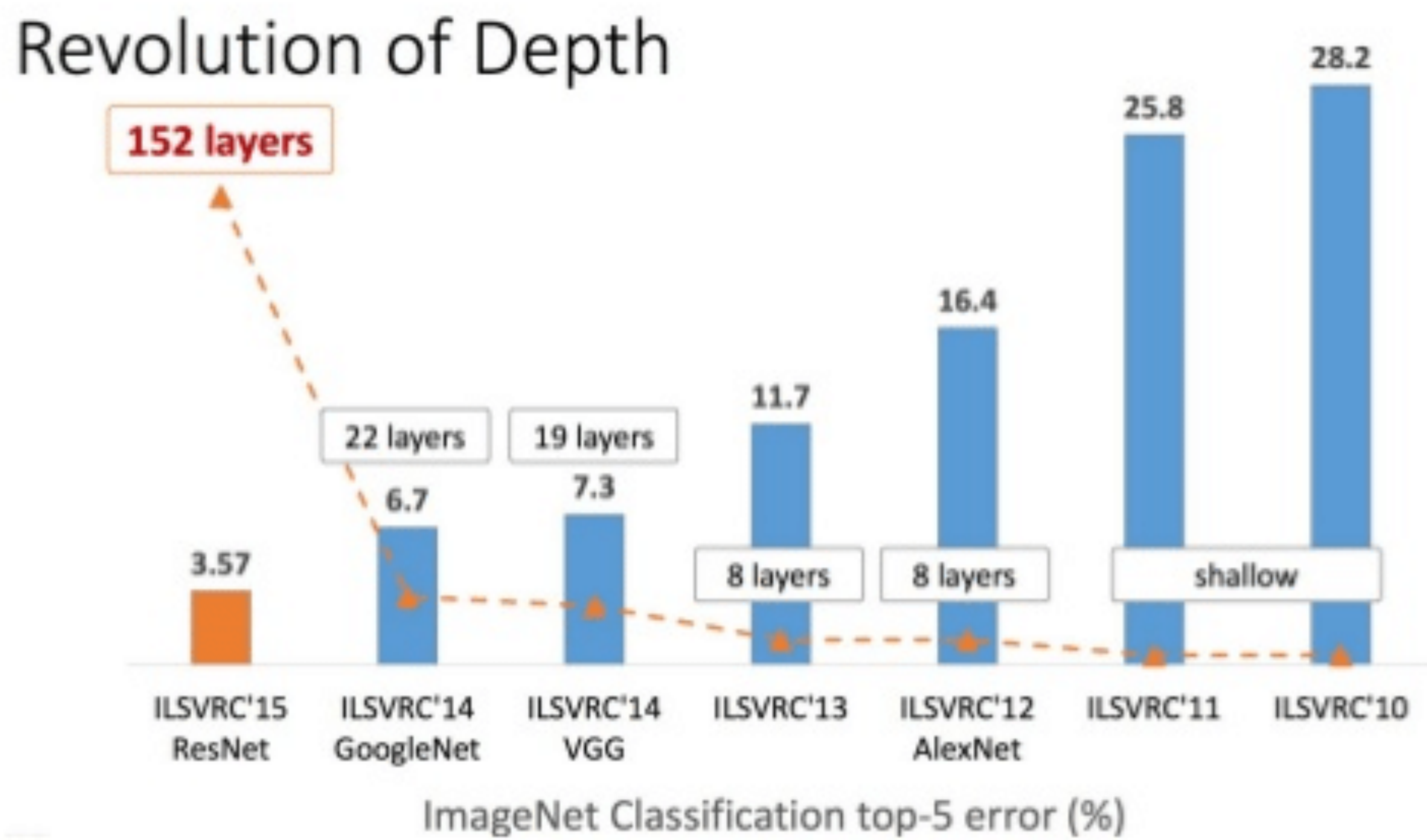
- AlexNet for image recognition



-8 layers, 650K neurons, 60M parameters
-Max pooling, ReLU nonlinearity
-1.2M training images of 1000 classes

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

CNN Architecture



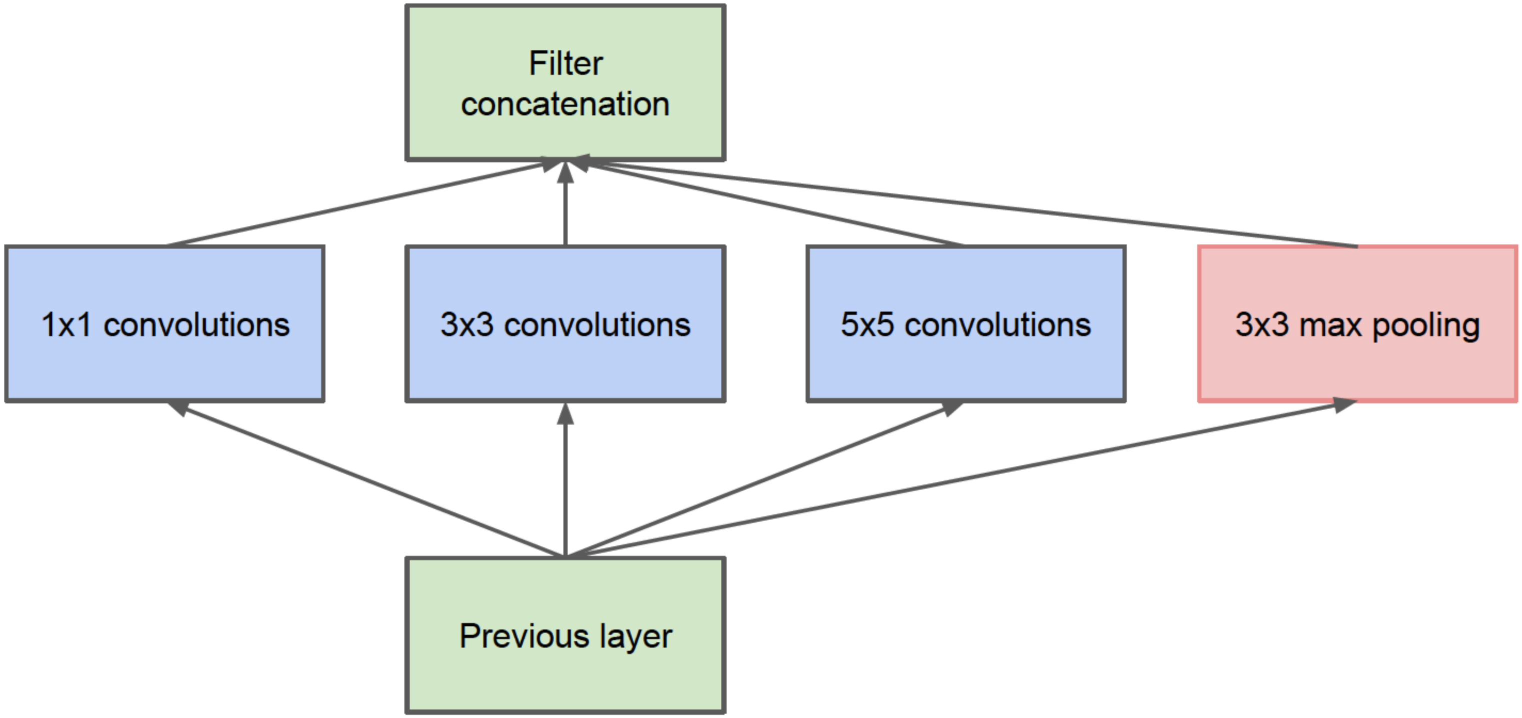
Inception module and GoogLeNet

- Filter size: hyperparameter
- What is the right filter size?
- Inception module
  - Use filters of different size in the same layer
  - Concatenate all the filter results as output
  - Use 1x1 convolution to reduce complexity
  - Increase width and depth of the network
  - All convolutions use ReLU, including 1x1 convolution for reduction / projection

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, “Going Deeper with Convolutions,” CVPR 2015

Inception module and GoogLeNet

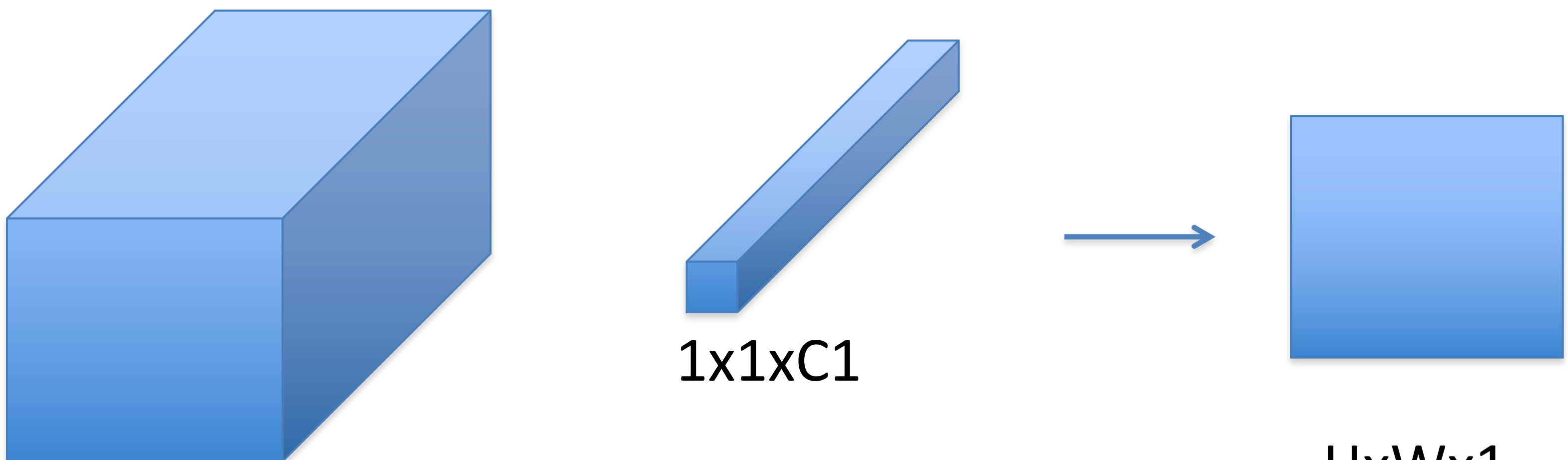
- Inception module, naïve version



Computation expensive: 28x28x192 input, need 5x5x192 filtering

Inception module and GoogLeNet

- 1x1 convolution for dimensionality reduction: reduce number of channels



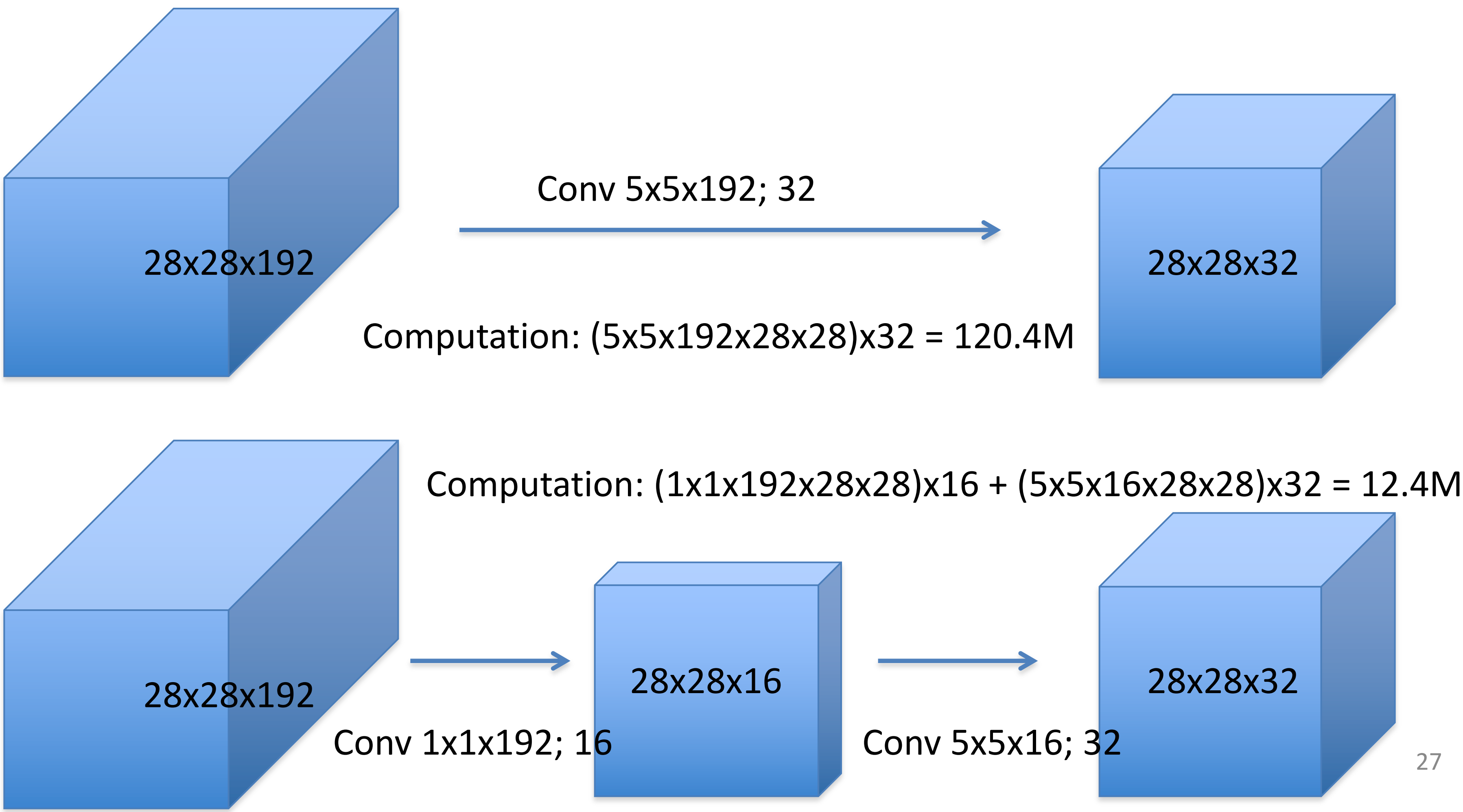
[Lin et al.; 2014]

-With C2 filters (1x1xC1 each), output HxWxC2; usually C2 < C1

-Followed by non-linear as in ordinary convolution

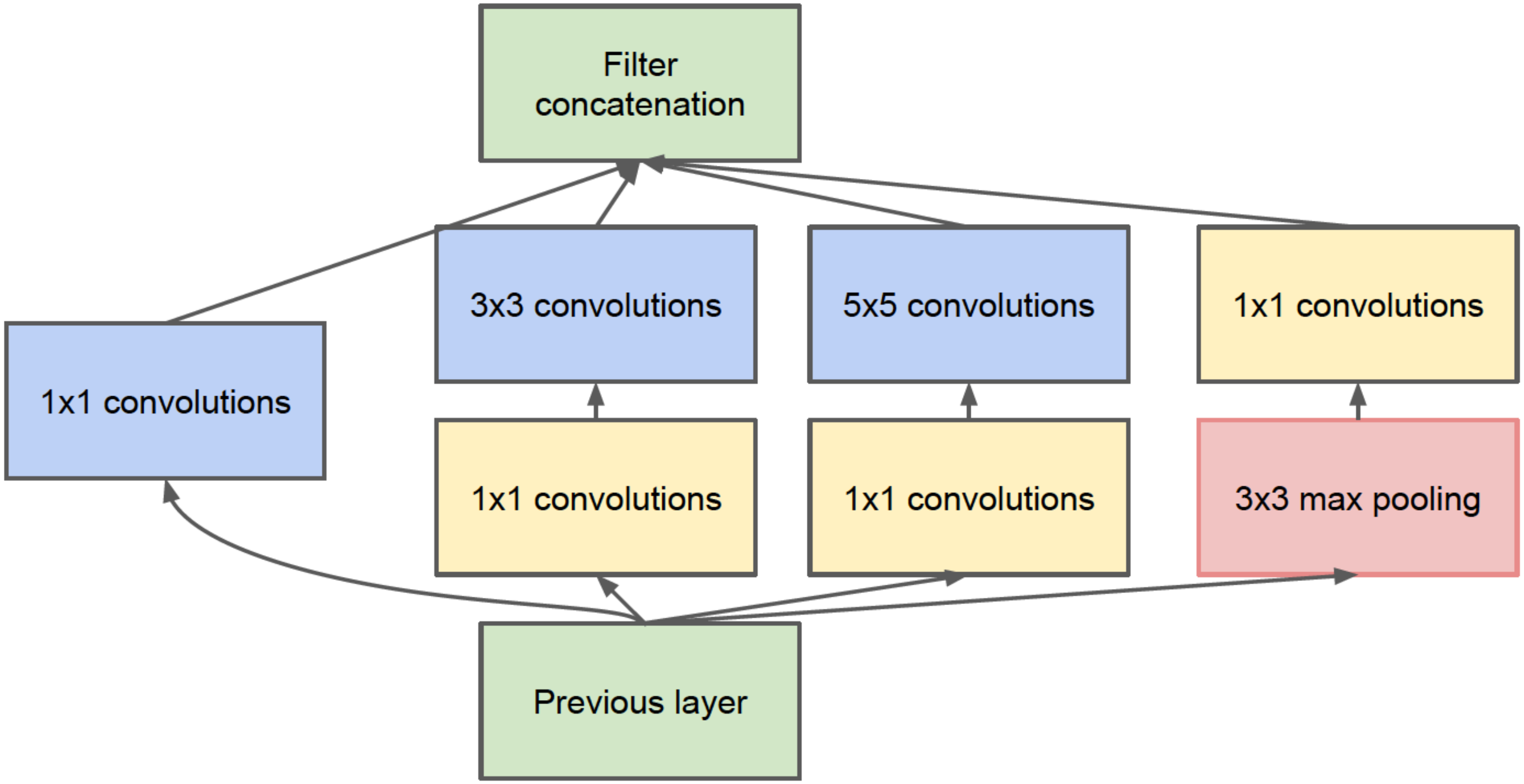
Inception module and GoogLeNet

- 1x1 convolution for dimensionality reduction:



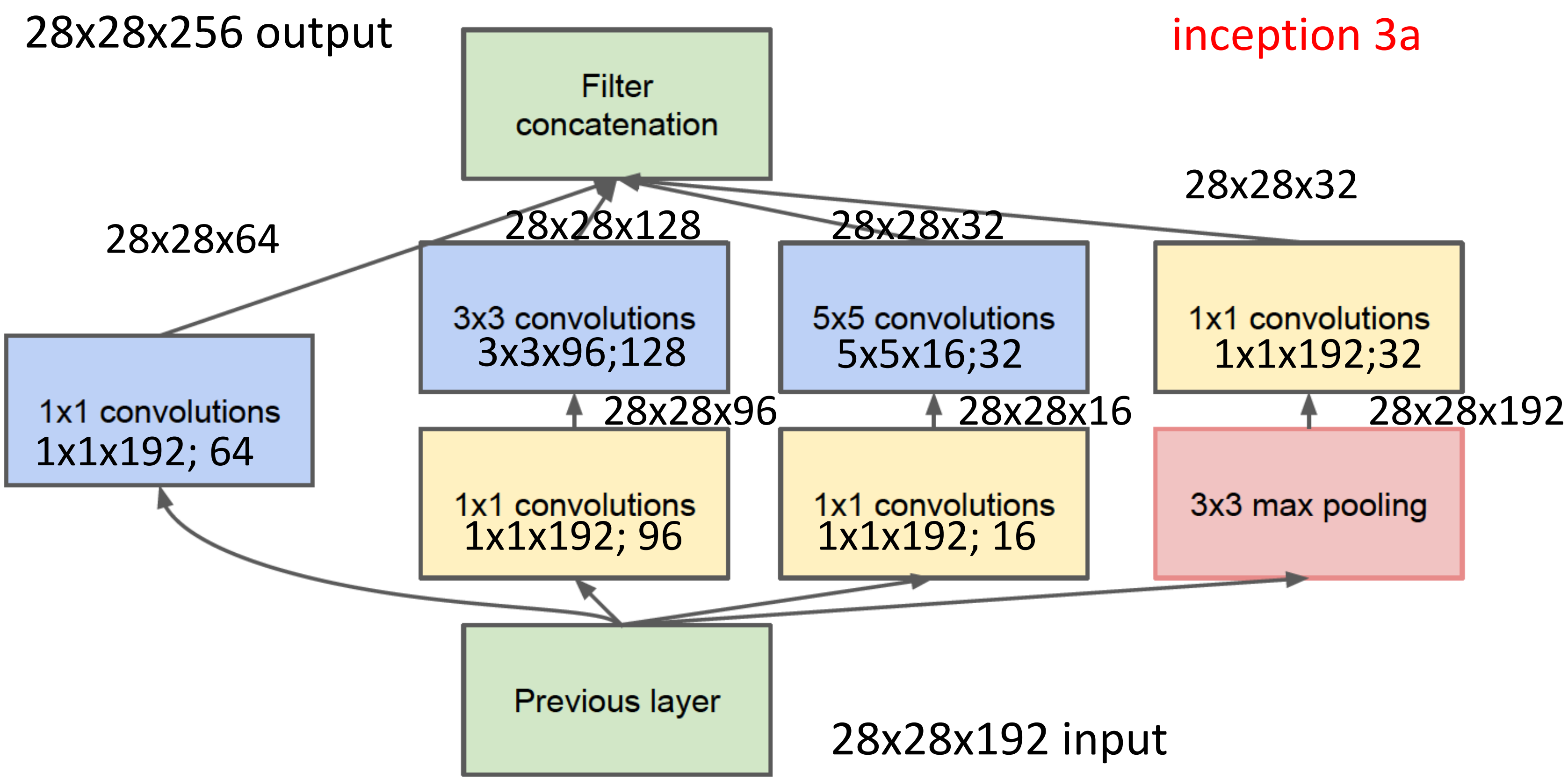
Inception module and GoogLeNet

- Inception module with dimensionality reduction



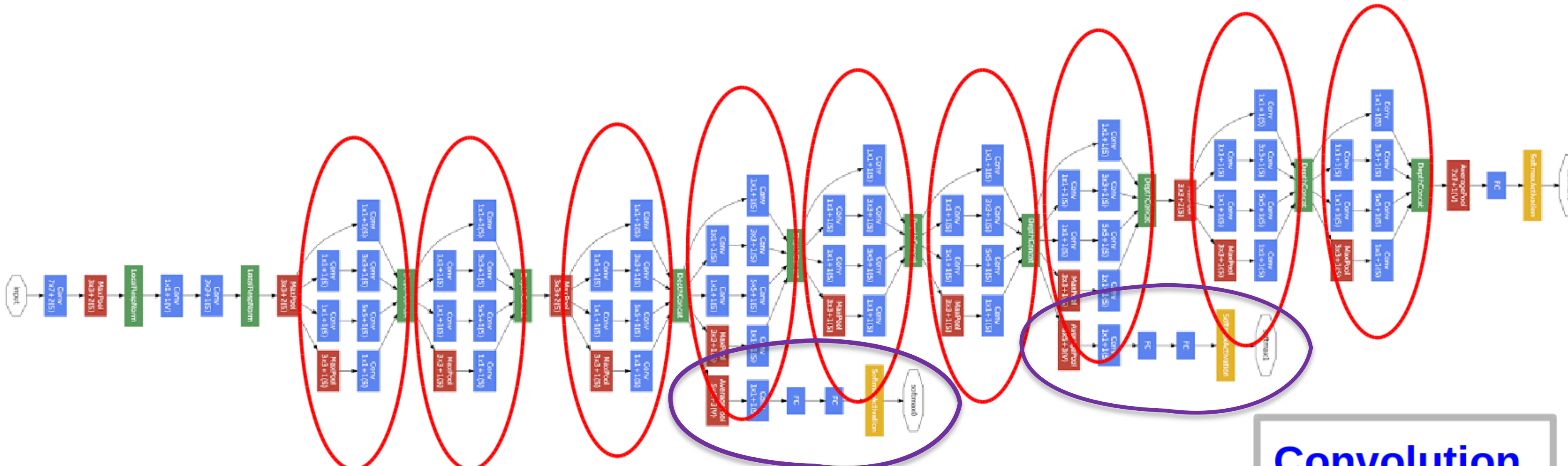
Inception module and GoogLeNet

- Inception module with dimensionality reduction:



Inception module and GoogLeNet

- GoogLeNet



Convolution
Pooling
Softmax
Concat/Normalize

Auxiliary networks:
added during training to avoid vanishing gradients. During training, their loss gets added to the total loss of the network with a discount weight (auxiliary classifiers were weighted by 0.3).
At inference time, these auxiliary networks are discarded.

Inception module and GoogLeNet

Total: 27 layers (dropout, softmax not count)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture.

Number of 1x1 filters in the reduction layer before 3x3 or 5x5 convolution

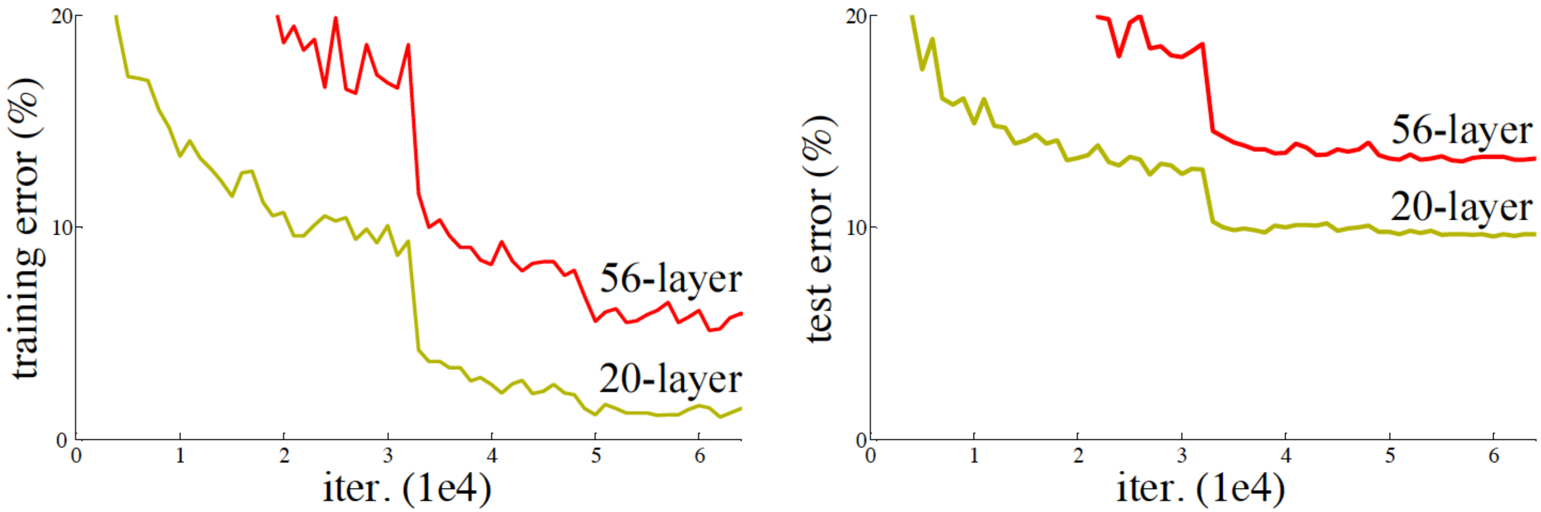
Number of 1x1 filters after max pooling

Number of 5x5 filters

Shortcut connection and Resnet

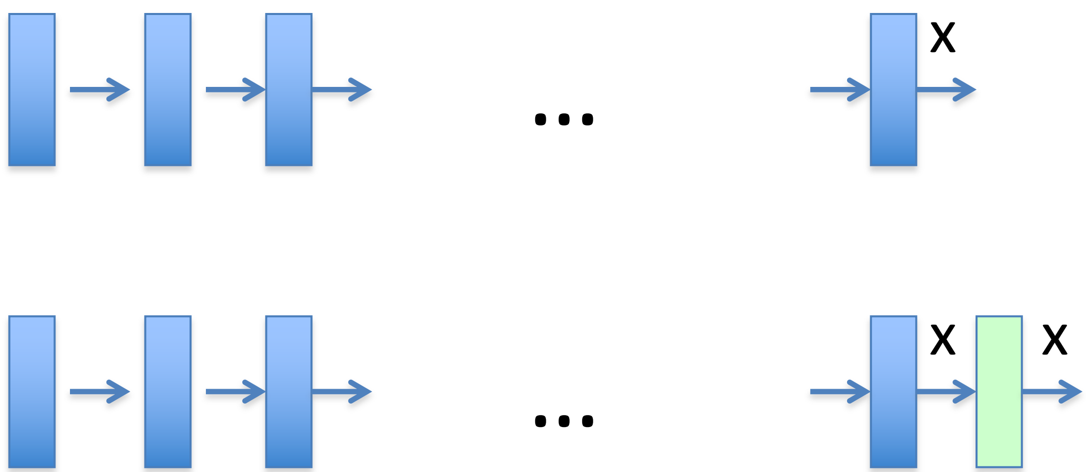
- Challenges of going deeper
  - Vanishing gradient: gradient is backpropagated to earlier layers, repeated multiplication may make the gradient very small
  - Overfitting: good in training, bad in testing
  - Difficult to learn an identity mapping in a deep architecture

Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error



Shortcut connection and Resnet

- Challenges of going deeper
  - Vanishing gradient: gradient is backpropagated to earlier layers, repeated multiplication may make the gradient very small
  - Overfitting: good in training, bad in testing
  - Difficult to learn an identity mapping in a deep architecture

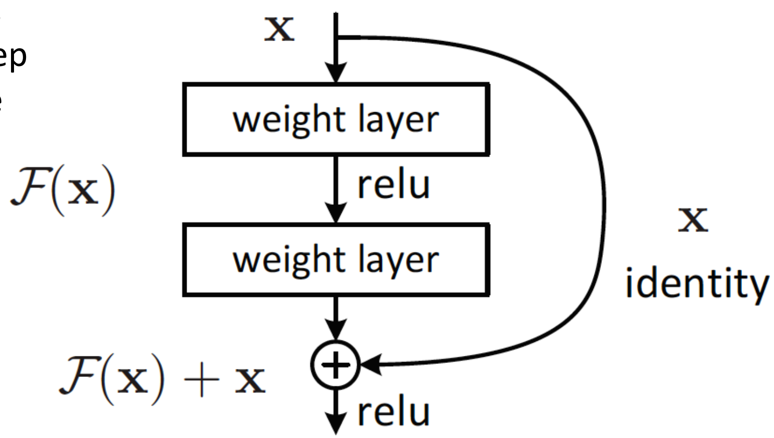


Adding an identity mapping layer would have no worse performance -> difficulty in learning an identity mapping results in poorer performance when going deeper

Shortcut connection and Resnet

- Deep residual learning
  - Learn the residual mapping instead of the entire mapping

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016



$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

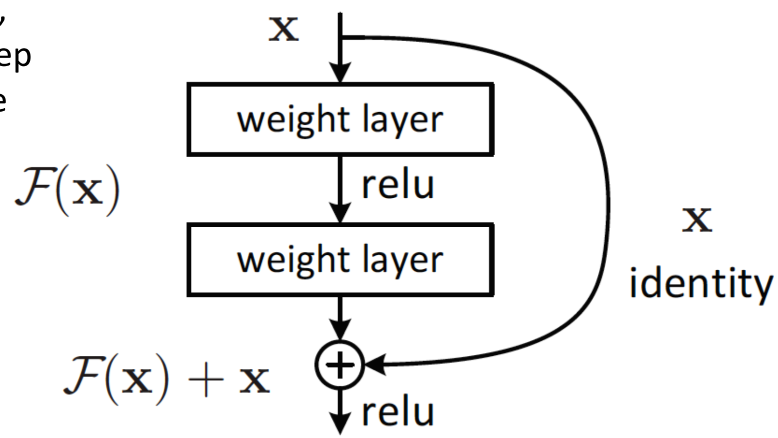
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

Element-wise addition, channel by channel

Shortcut connection and Resnet

- Deep residual learning
  - Learn the residual mapping instead of the entire mapping

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016



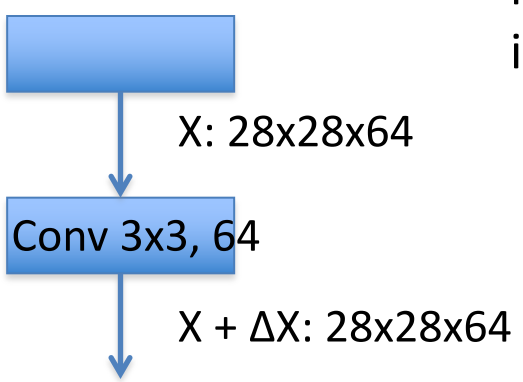
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

$$\mathcal{F} = W_2 \sigma(W_1 \mathbf{x})$$

Optimal mapping is close to an identity mapping, residual learning is a construction to ease the learning

Understanding Residual Learning

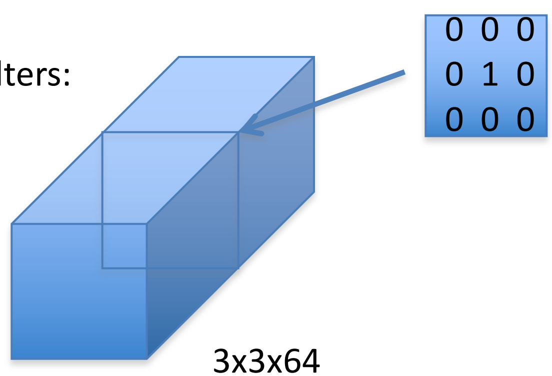
- Deep residual learning
  – Learn the residual mapping instead of the entire mapping



Want ΔX to be small, so that the output feature maps would not cause dramatic change (degrade) in performance

- Incremental approach to build complex DNN
- Need W ≈ 1, i.e., identity mapping
- But hard to learn with data-driven approach
- An issue with optimization (non-convex, high dimensional loss function)

One of the 64 filters:



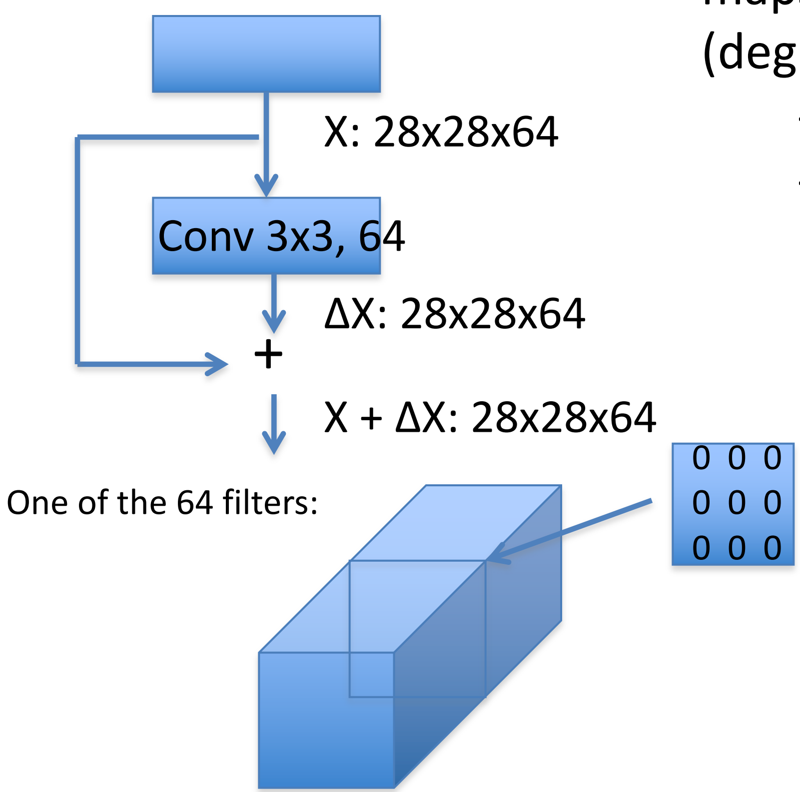
All zero in other slices

Understanding Residual Learning

- Deep residual learning
  - Learn the residual mapping instead of the entire mapping

Want ΔX to be small, so that the output feature maps would not cause dramatic change (degrade) in performance

- With skip connection, need W ≈ 0
- Easy to learn with regularization on W

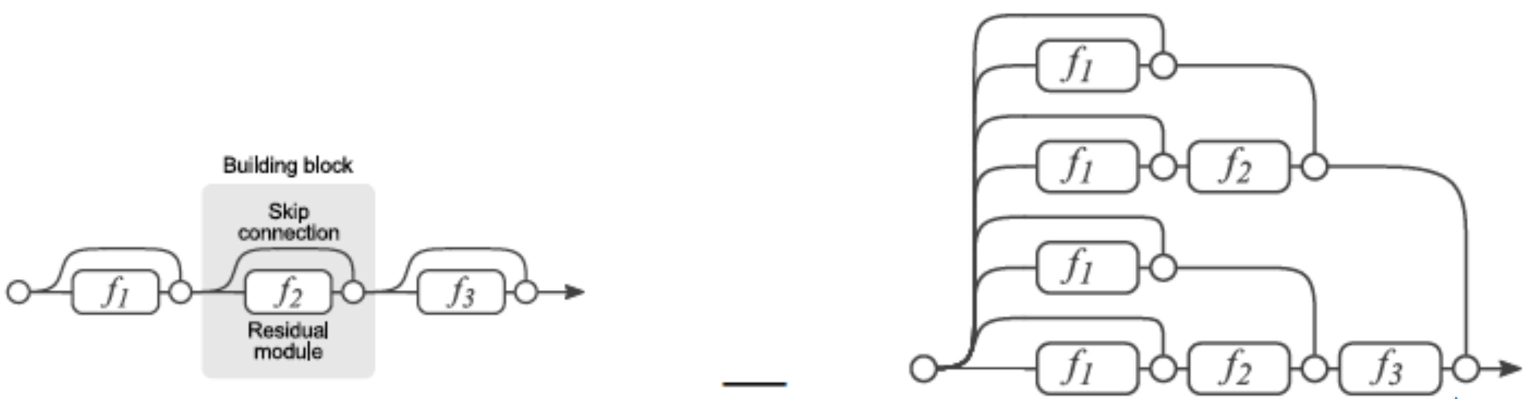


"We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers."

Understanding Residual Learning

- Another reasoning: Ensembles of networks

-Residue networks can be viewed as a collection of many paths, instead of a single ultra-deep network



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

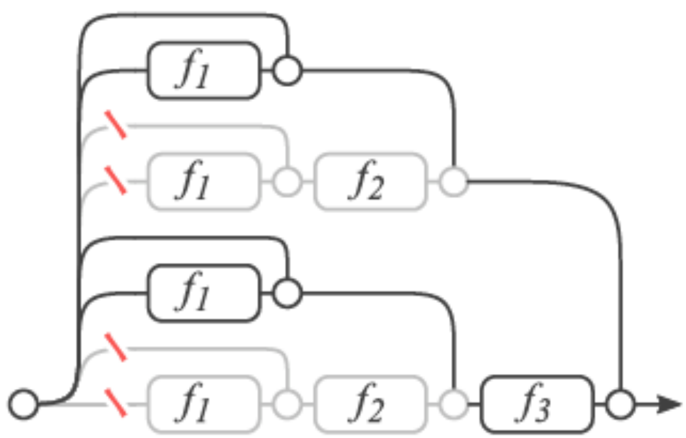
Averaging many networks

Andreas Veit, Michael Wilber, Serge Belongie, “Residual Networks Behave Like Ensembles of Relatively Shallow Networks,” NIPS 2016

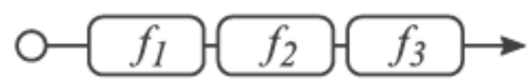
Understanding Residual Learning

- Another reasoning: Ensembles of networks

-These paths do not strongly depend on each other, even thought they are trained jointly
-Exhibit ensemble-like behavior: overall performance correlates smoothly with the number of valid paths



(a) Deleting f_2 from unraveled view



(b) Ordinary feedforward network

Figure 2: Deleting a layer in residual networks at test time (a) is equivalent to zeroing half of the paths. In ordinary feed-forward networks (b) such as VGG or AlexNet, deleting individual layers alters the only viable path from input to output.

Andreas Veit, Michael Wilber, Serge Belongie, “Residual Networks Behave Like Ensembles of Relatively Shallow Networks,” NIPS 2016

Understanding Residual Learning

- Another reasoning: Ensembles of networks

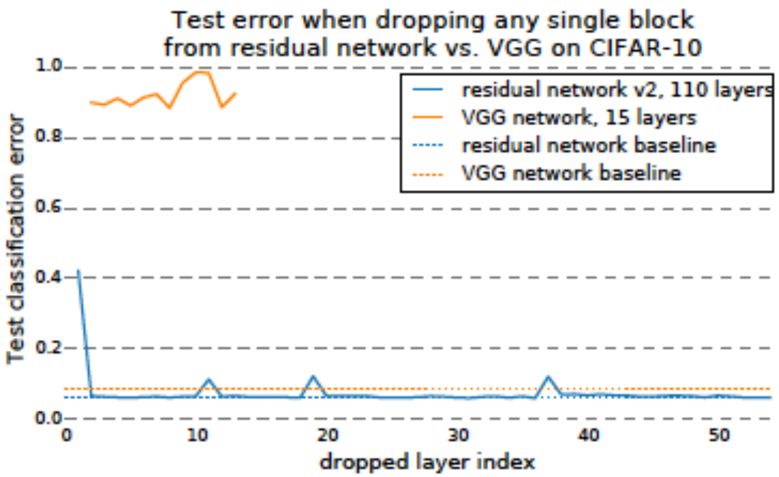


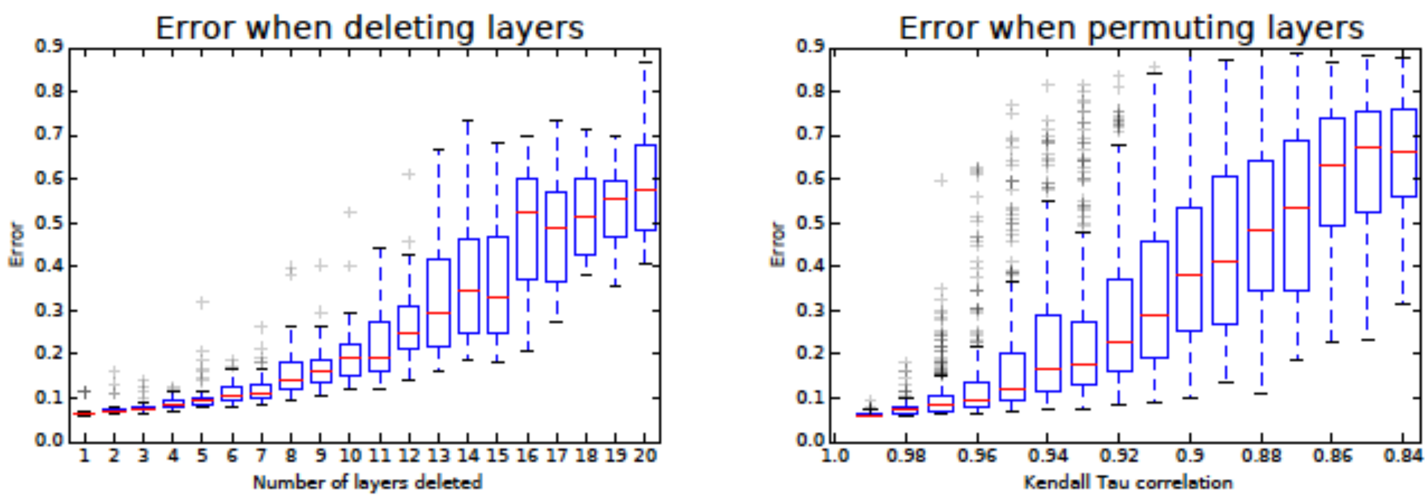
Figure 3: Deleting individual layers from VGG and a residual network on CIFAR-10. VGG performance drops to random chance when any one of its layers is deleted, but deleting individual modules from residual networks has a minimal impact on performance. Removing downsampling modules has a slightly higher impact.

-These paths do not strongly depend on each other, even they are trained jointly
-Exhibit ensemble-like behavior: overall performance correlates smoothly with the number of valid paths

Andreas Veit, Michael Wilber, Serge Belongie, "Residual Networks Behave Like Ensembles of Relatively Shallow Networks," NIPS 2016

Understanding Residual Learning

- Another reasoning: Ensembles of networks



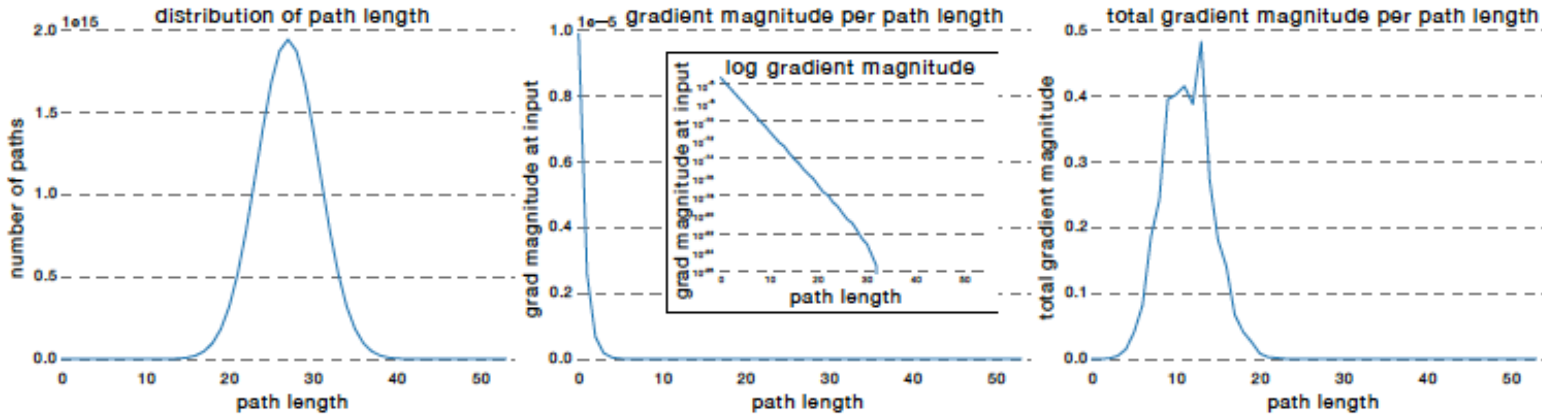
(a) (b)

Figure 5: (a) Error increases smoothly when randomly deleting several modules from a residual network. (b) Error also increases smoothly when re-ordering a residual network by shuffling building blocks. The degree of reordering is measured by the Kendall Tau correlation coefficient. These results are similar to what one would expect from ensembles.

Andreas Veit, Michael Wilber, Serge Belongie, “Residual Networks Behave Like Ensembles of Relatively Shallow Networks,” NIPS 2016

Understanding Residual Learning

- Another reasoning: Ensembles of networks



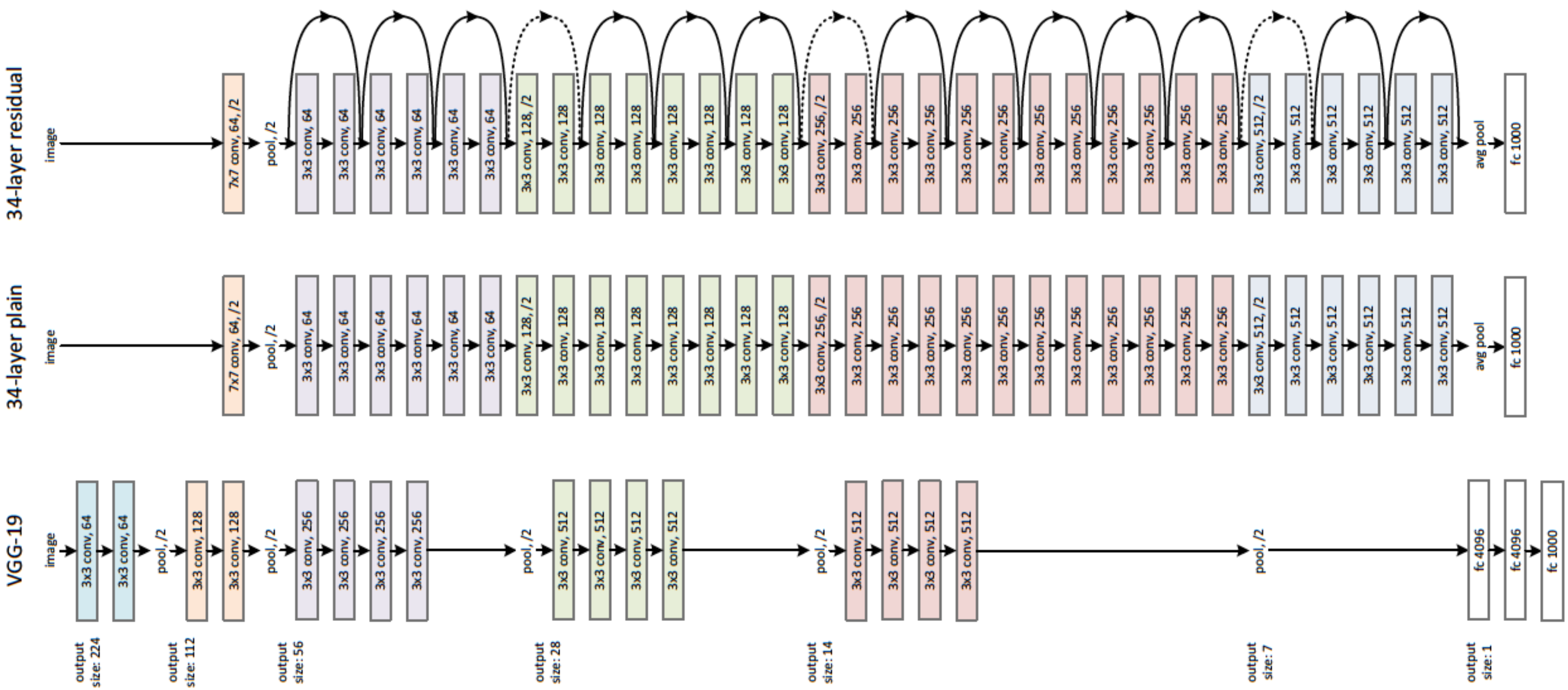
(a) (b) (c)

Figure 6: How much gradient do the paths of different lengths contribute in a residual network? To find out, we first show the distribution of all possible path lengths (a). This follows a Binomial distribution. Second, we record how much gradient is induced on the first layer of the network through paths of varying length (b), which appears to decay roughly exponentially with the number of modules the gradient passes through. Finally, we can multiply these two functions (c) to show how much gradient comes from all paths of a certain length. Though there are many paths of medium length, paths longer than ~20 modules are generally too long to contribute noticeable gradient during training. This suggests that the effective paths in residual networks are relatively shallow.

Andreas Veit, Michael Wilber, Serge Belongie, “Residual Networks Behave Like Ensembles of Relatively Shallow Networks,” NIPS 2016

Shortcut connection and Resnet

- Resnet: stack of deep residual learning modules



Other variants: ResNet-50/101/152

Today’s class

- CNN architectures:
  - LeNet
  - AlexNet
  - GoogleLeNet (Inception)
  - ResNet

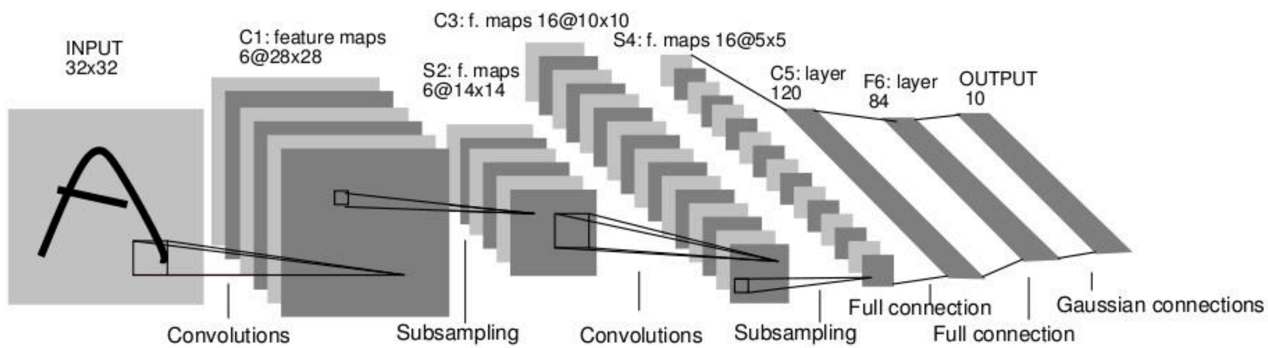
- **Backpropagation**

Learn W using loss function L(W)

- Determine W with the min loss function

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

N training samples

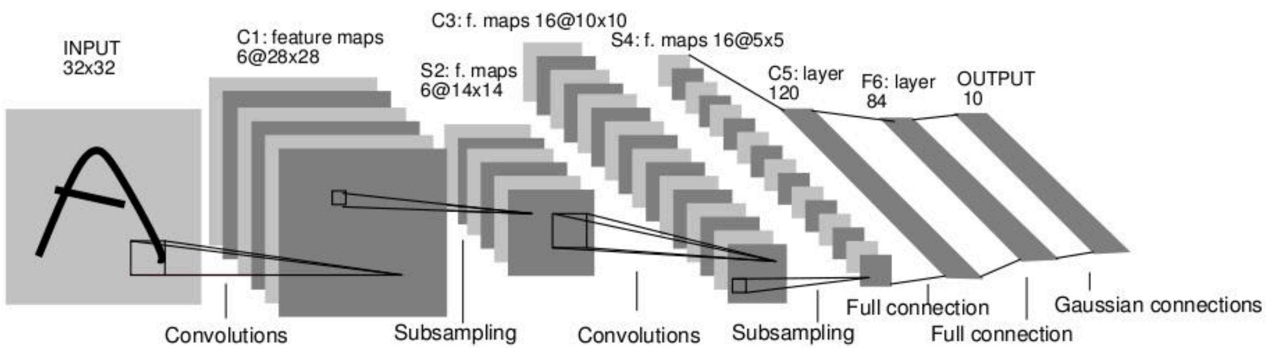


Learn W using loss function L(W)

- Determine W with the min loss function

$$L = \frac{1}{N}\sum_i L_i + \lambda R(W)$$

N training samples



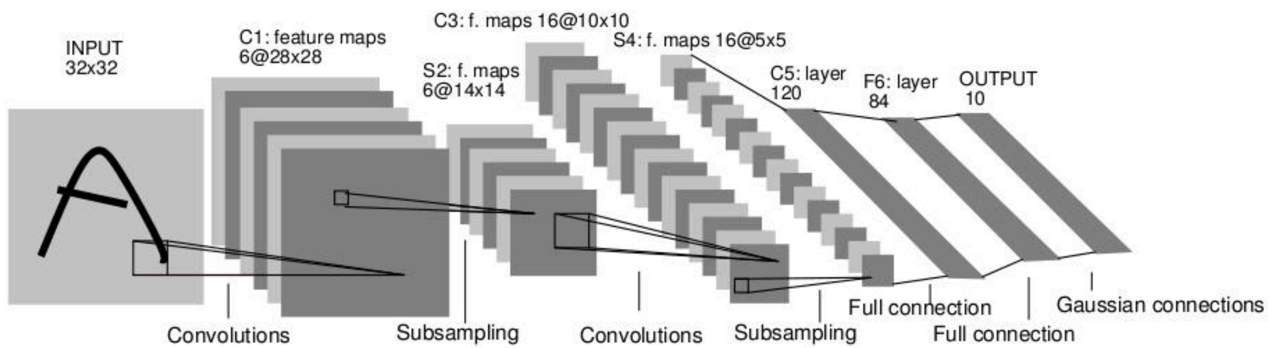
- Start from a random W, iteratively improve W (reduce L(W)): Gradient descent
- Note: L(W) = L(W; (x_1,y_1), (x_2,y_2), ...(x_i,y_i)...(x_N,y_N))

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$L(W) = L(w_1, w_2, ...w_l...)$$

$$W' = W - \gamma \nabla L \qquad \text{Gradient descent}$$



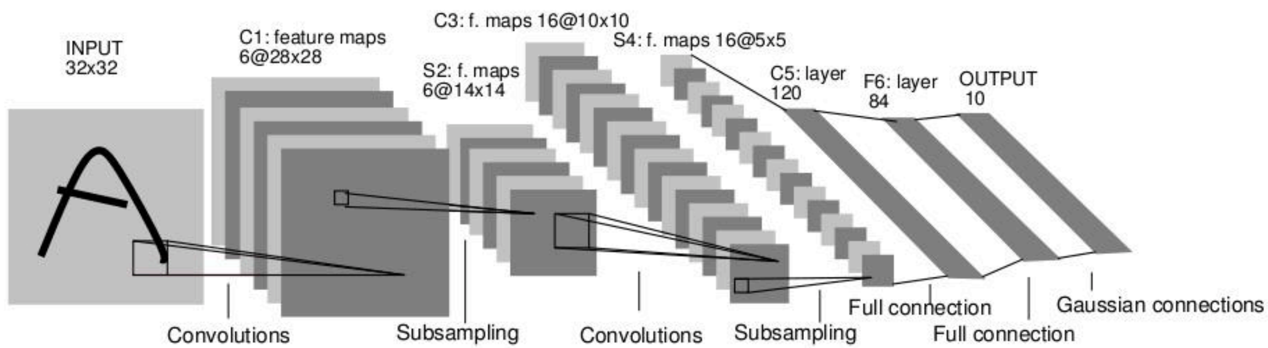
$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$L(W) = L(w_1, w_2, ...w_l...)$$

$$W' = W - \gamma \nabla L$$ Gradient descent



$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

Find $\frac{\partial L}{\partial w_l}$ for every w_l

Learn W by gradient descent

- Update W by W+ΔW, using the gradient

$$w_l' = w_l - \gamma \frac{\partial L}{\partial w_l}$$

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

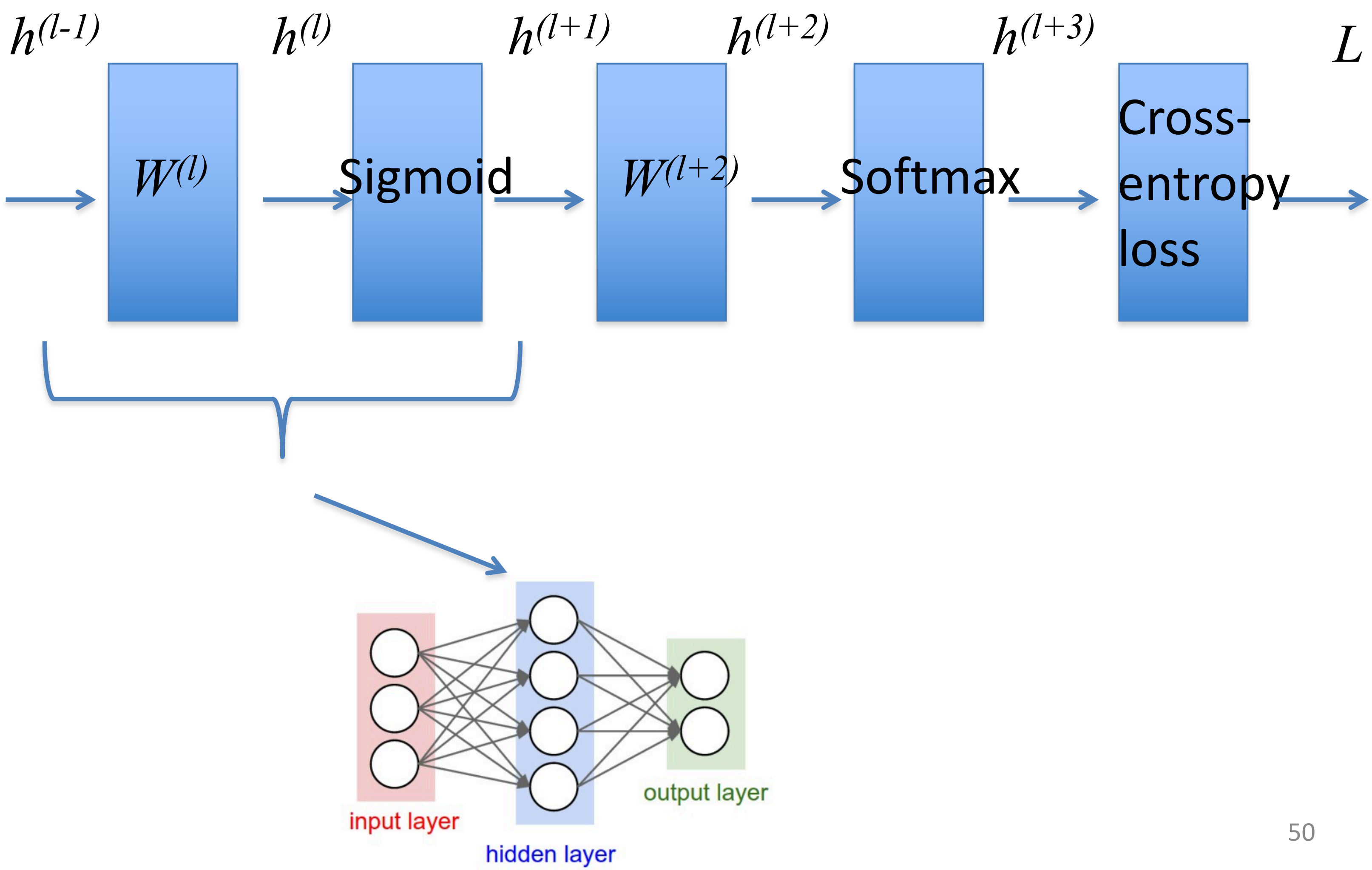
$$\frac{\partial L}{\partial w_l} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial w_l} + \lambda \frac{\partial R(W)}{\partial w_l}$$

- Sum gradients for all (partial) training samples for one w_l
- Make one update of W once we have **all the gradients**

Note: From now, here we refer to gradient of one training sample for simplicity

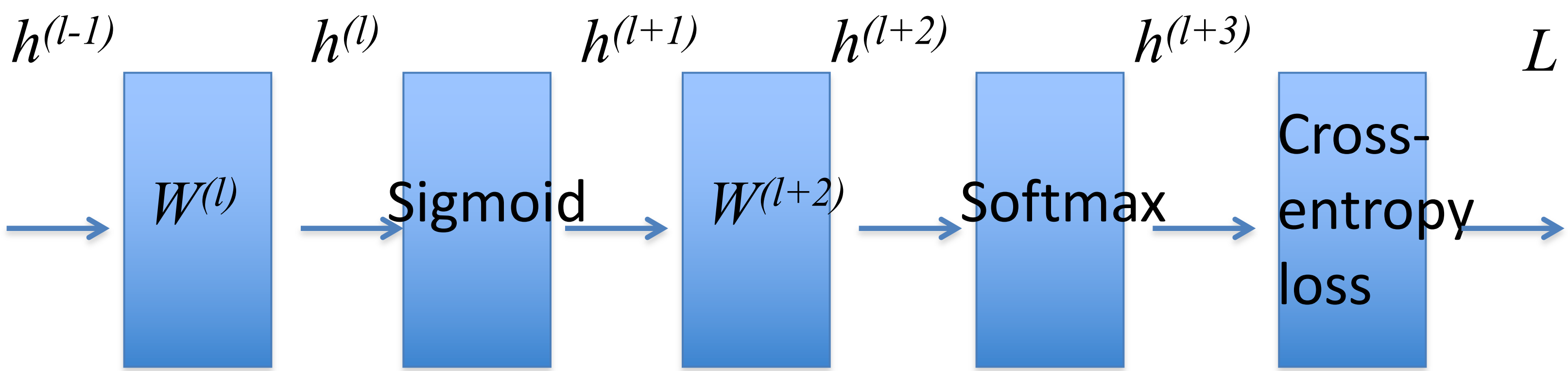
Compute the gradient

- Example



Compute the gradient

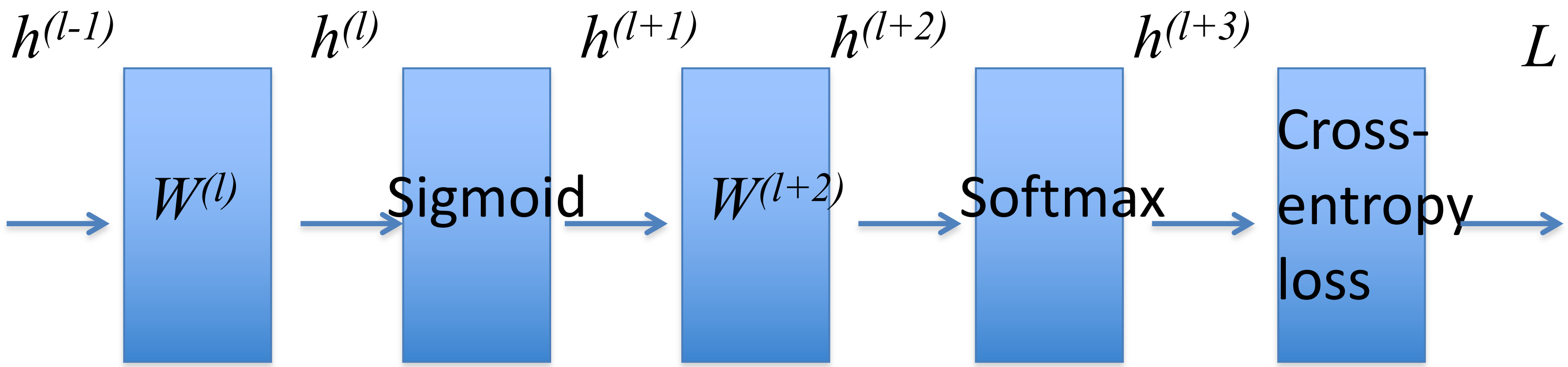
- Example



Approach 1: compute $L(W^{(l)})$, then differentiate

Compute the gradient

- Example

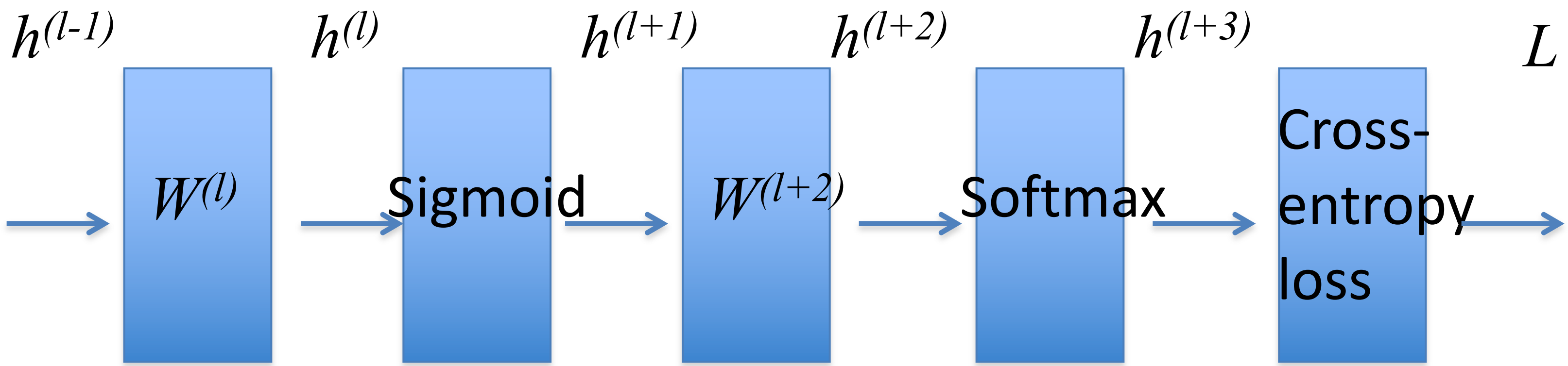


Approach 1: compute $L(W^{(l)})$, then differentiate

-Tedious task, e.g. for deep NN
-Not flexible; if some layer changes (Sigmoid -> ReLU), need to re-derive again

Compute the gradient

- Example



Approach 2: back propagation

-Assume we have $\frac{\partial L}{\partial h^{(l)}}$

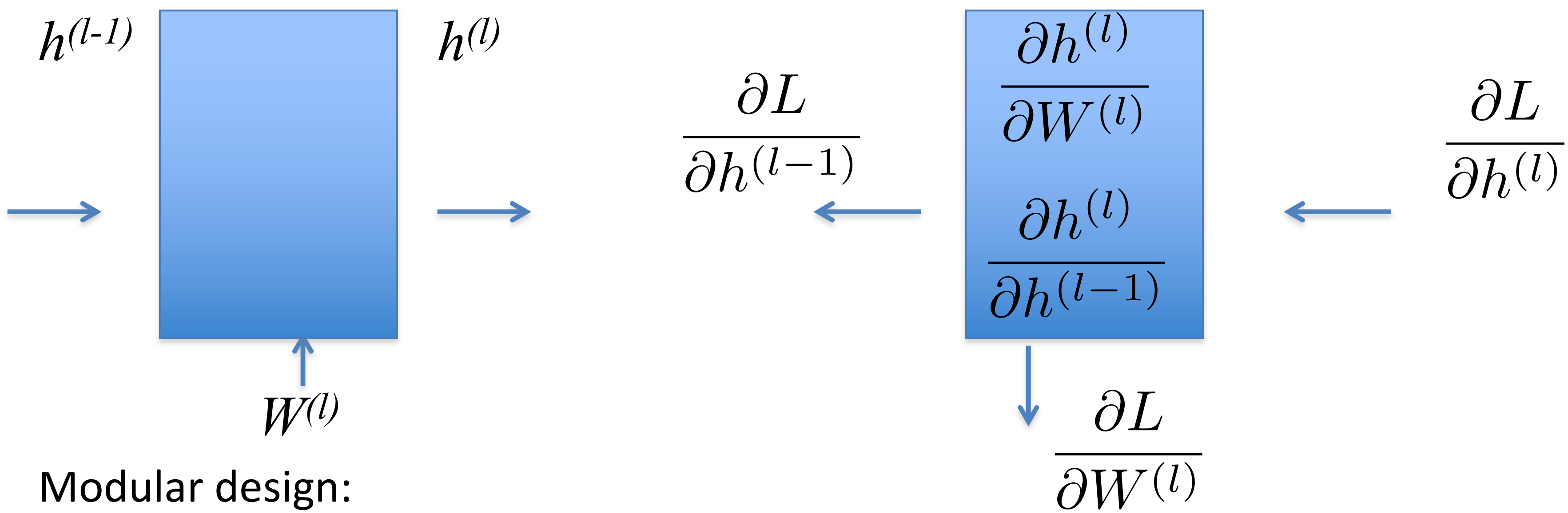
$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial W^{(l)}}$$

For grad descent

$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

Back prop



Modular design:

-Each module knows how to compute local gradients

-Multiply by the global gradient from upstream

-Generalize to other modules, e.g. ReLU

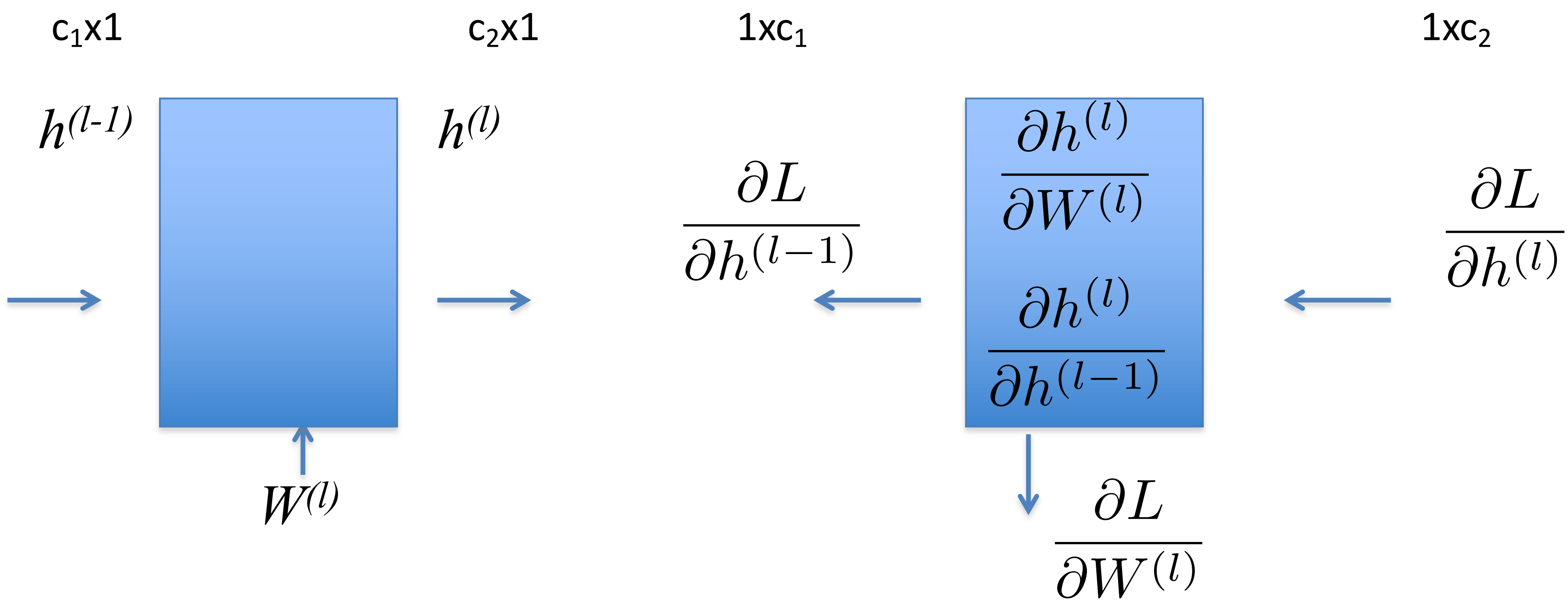
$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial W^{(l)}}$$

For grad descent

$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h^{(l)}}{\partial h^{(l-1)}}$$

For back prop

Back prop: fully connected



$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} W^{(l)}$$

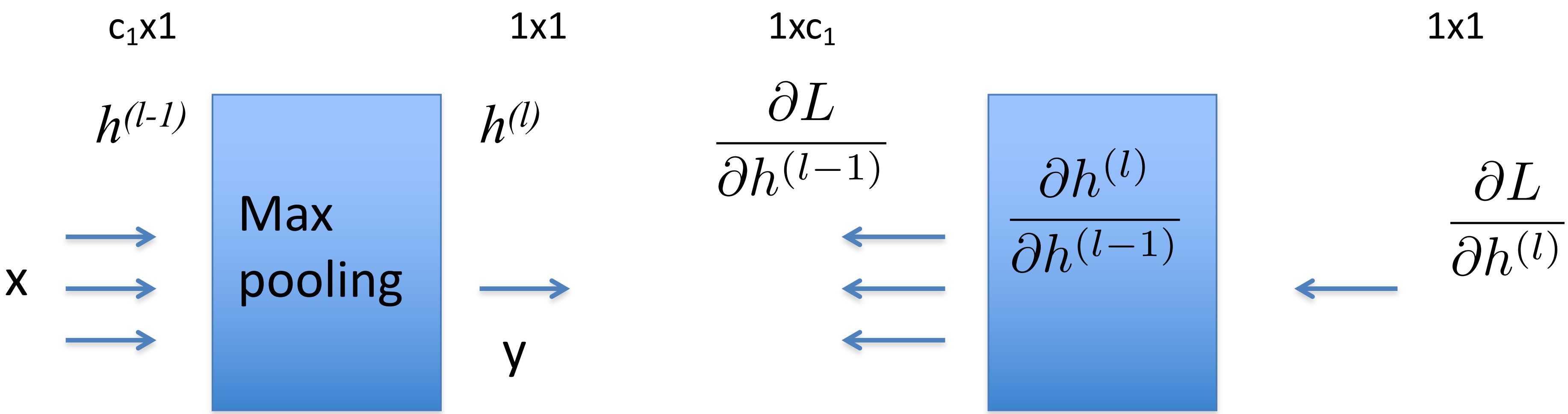
$$\frac{\partial L}{\partial \mathbf{w}_i^{(l)}} = \frac{\partial L}{\partial h_i^{(l)}} [h^{(l-1)}]^T$$

Recall:

$h^{(l)} = W^{(l)} h^{(l-1)}$

$\mathbf{w}_i^{(l)}$: i-th row of $W^{(l)}$

Back prop: max pooling



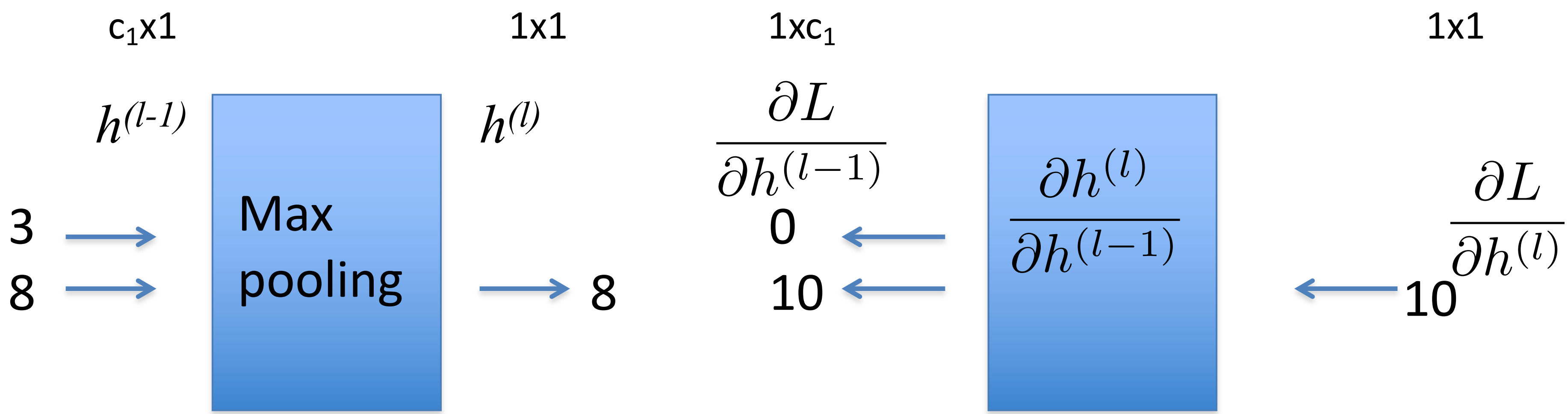
If x is max, y=x, dy/dx = 1
If x is not max, y and x are independent, dy/dx=0

$$\frac{\partial h^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} \text{ is max}) \qquad \frac{\partial L}{\partial h_i^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \mathbf{I}(h_i^{(l-1)} \text{ is max})$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Only 1 branch has gradient, and
gradient goes to the max input branch

Back prop: max pooling



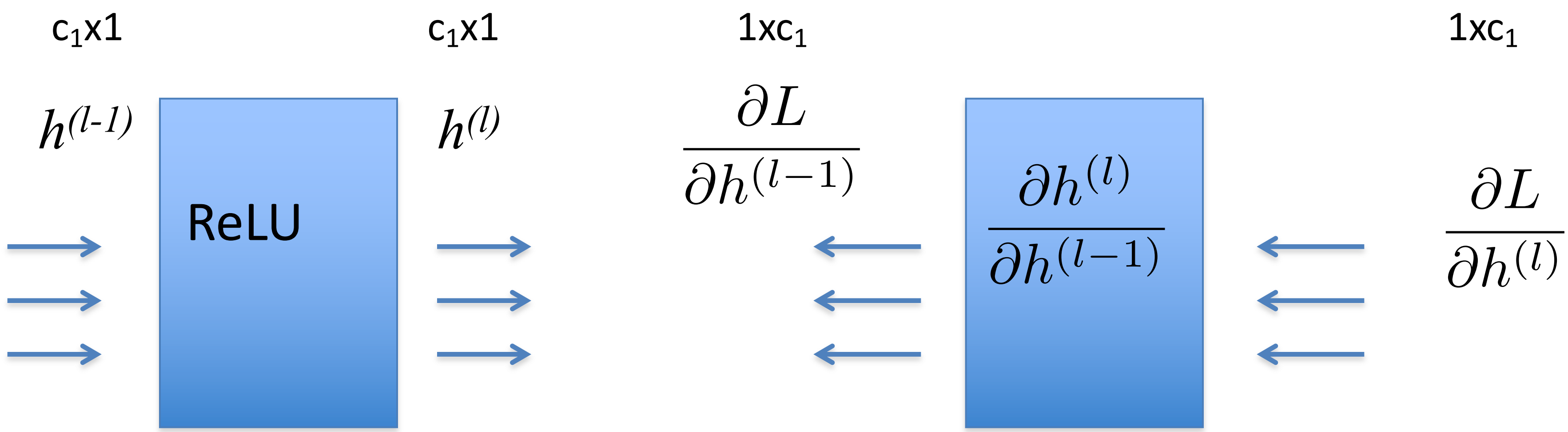
If x is max, y=x, dy/dx = 1
If x is not max, y and x are independent, dy/dx=0

$$\frac{\partial h^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} \text{ is } \max) \qquad \frac{\partial L}{\partial h_i^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \mathbf{I}(h_i^{(l-1)} \text{ is } \max)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Only 1 branch has gradient, and
gradient goes to the max input branch

Back prop: ReLU

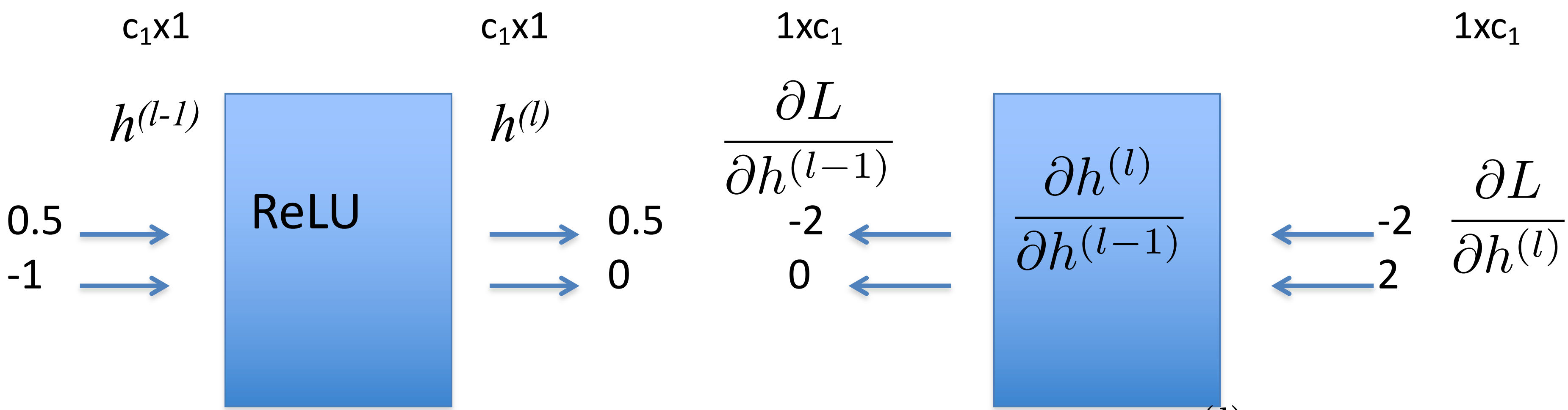


$$h_i^{(l)} = \max(0, h_i^{(l-1)}) \qquad \frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} > 0)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Gate: gradient can or cannot pass through

Back prop: ReLU



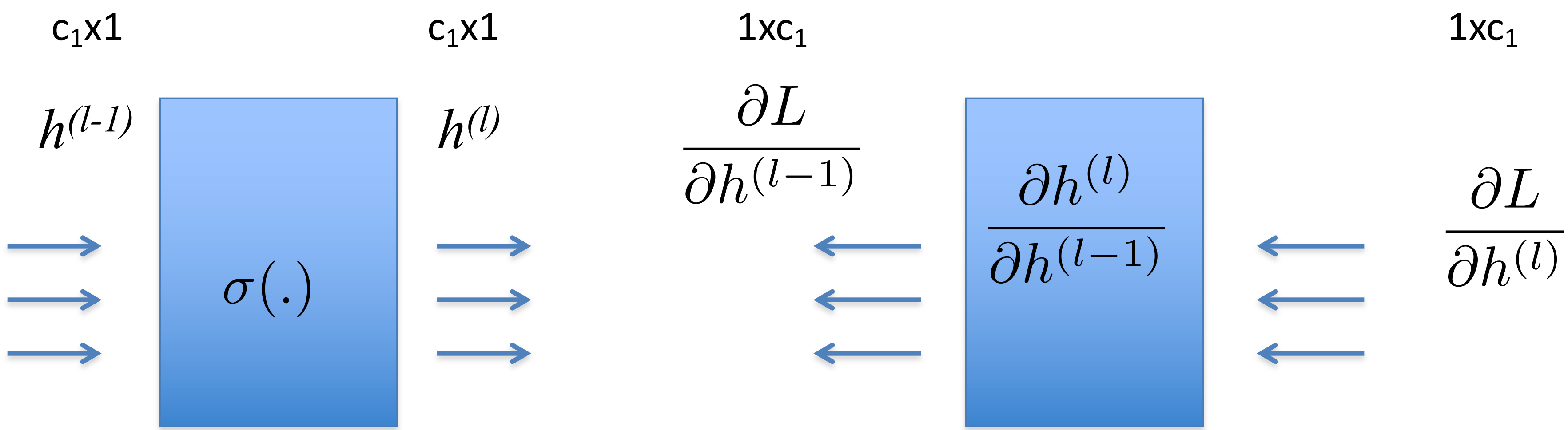
$$\frac{\partial L}{\partial h^{(l-1)}} = \frac{\partial L}{\partial h^{(l)}} \frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}}$$

$$h_i^{(l)} = \max(0, h_i^{(l-1)}) \qquad \frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \mathbf{I}(h_i^{(l-1)} > 0)$$

$\mathbf{I}$(c) = 1 if c is true, 0 otherwise

Gate: gradient can or cannot pass through

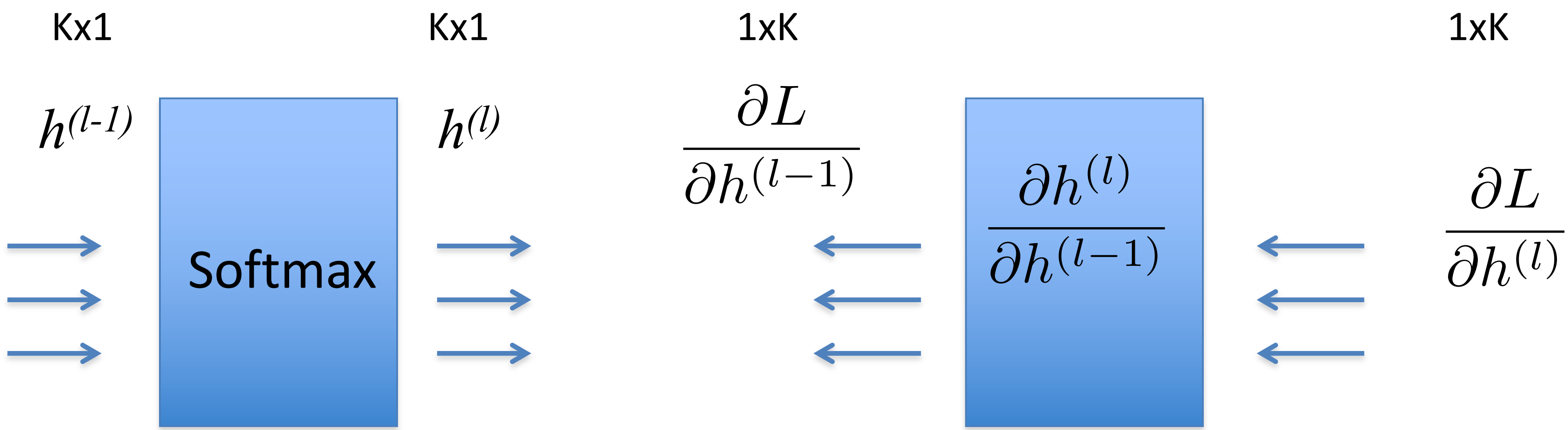
Back prop: Sigmoid



$$h_i^{(l)} = \sigma(h_i^{(l-1)})$$

$$\frac{\partial h_i^{(l)}}{\partial h_i^{(l-1)}} = \sigma(h_i^{(l-1)})(1 - \sigma(h_i^{(l-1)}))$$

Back prop: Softmax

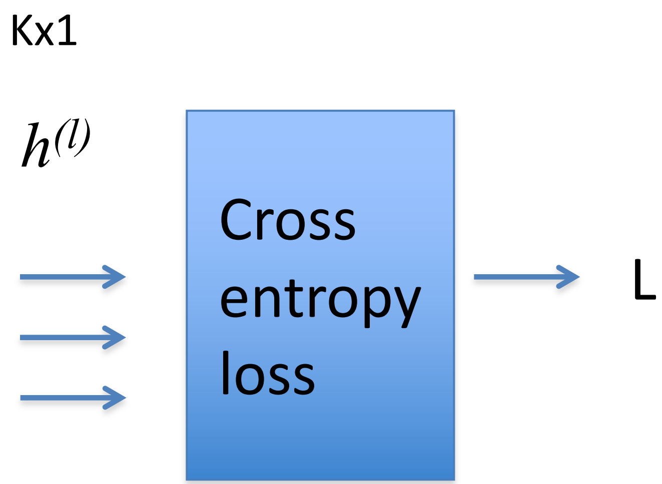


$\frac{\partial h^{(l)}}{\partial h^{(l-1)}}$ is a KxK Jacobian matrix

$$\frac{\partial h_i^{(l)}}{\partial h_j^{(l-1)}} = h_i^{(l)}(1 - h_j^{(l)}) \qquad i = j$$

$$\frac{\partial h_i^{(l)}}{\partial h_j^{(l-1)}} = -h_i^{(l)} h_j^{(l)} \qquad i \neq j$$

Back prop: Cross entropy loss



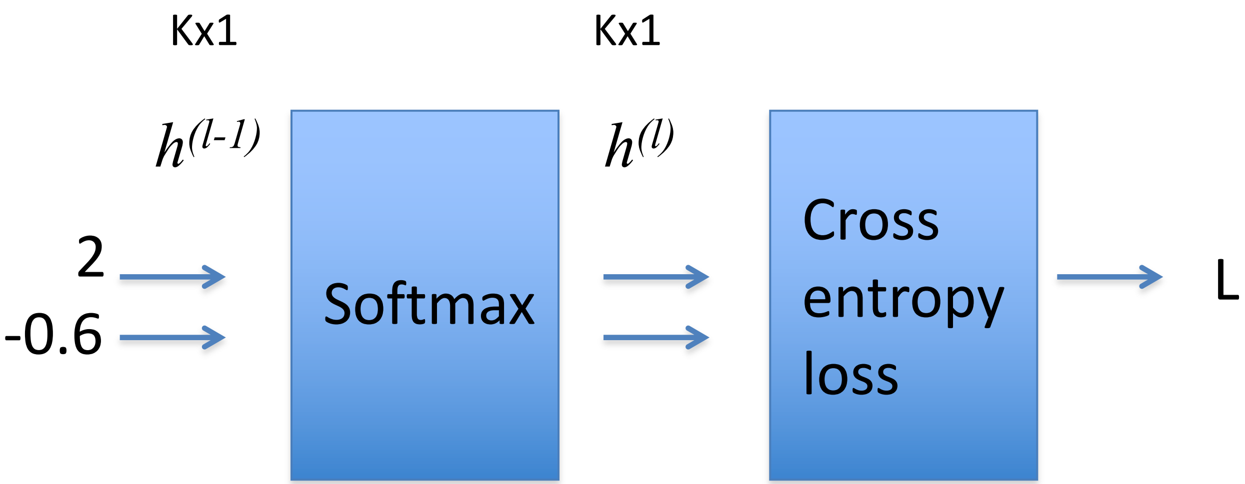
The y-th class is the ground-truth

$$L = -\log(h_y^{(l)})$$

$$\frac{\partial L}{\partial h^{(l)}} = [0, 0, ..., \frac{-1}{h_y^{(l)}}, ..., 0]$$

Back prop starts from the loss function

Back prop: Exercise



The first class is the ground-truth

Compute:

(i) L

(ii) $\frac{\partial L}{\partial h^{(l-1)}}$

(i) 0.07

(ii) [-0.0687 0.0687]

Today's class

- CNN architectures:
  - LeNet
  - AlexNet
  - GoogleLeNet (Inception)
  - ResNet
- Backpropagation

Next week's class

- Presentation of project idea and preliminary results by the teams (8 and 10 Nov)