The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2024

INTRODUCTION TO COMPUTER VISION

Atlas Wang
Associate Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

A Triumph of Deep Learning: 2012 - present

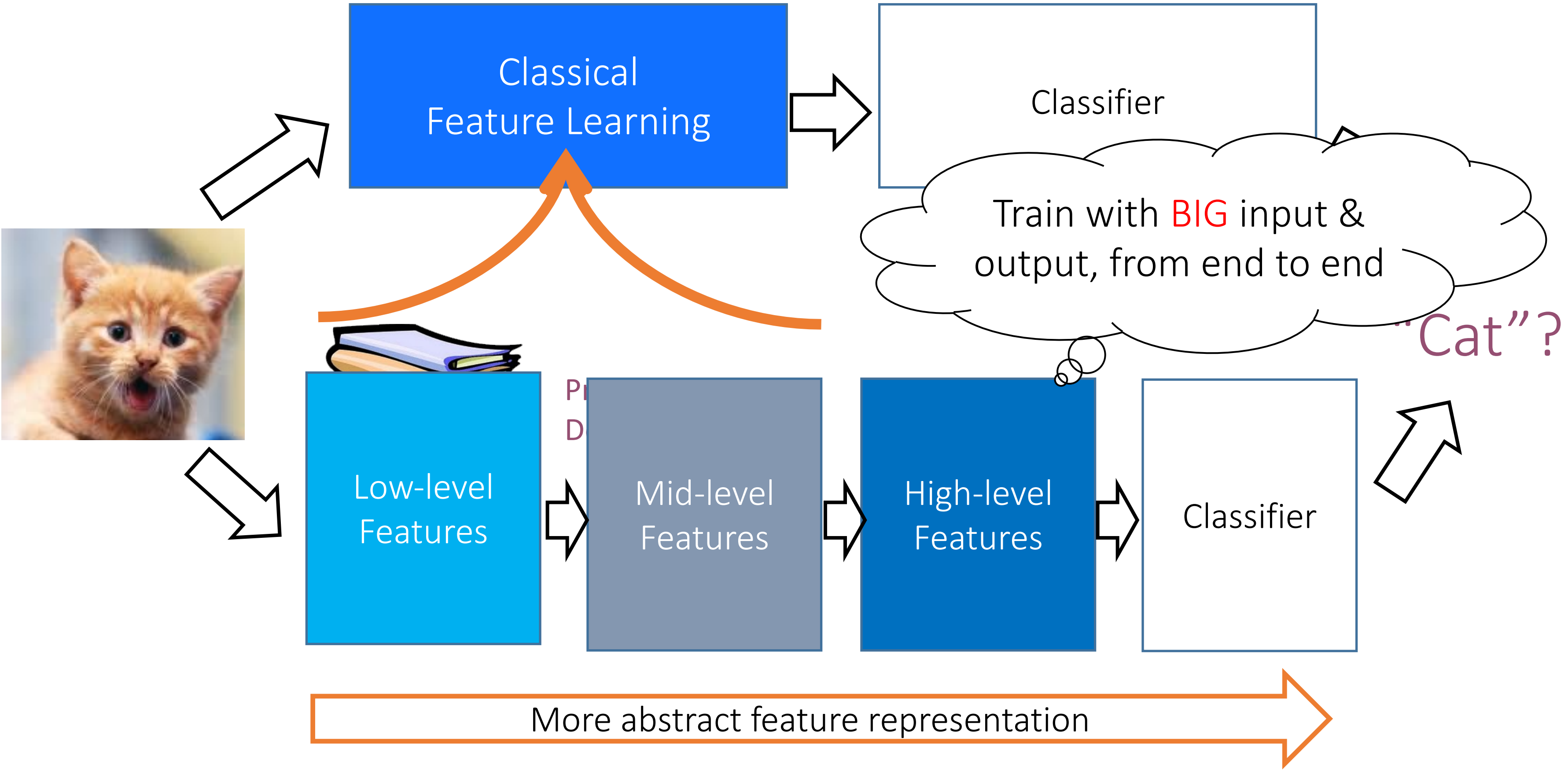
Top-performers in many tasks, over many domains

Speech Recognition

NLP

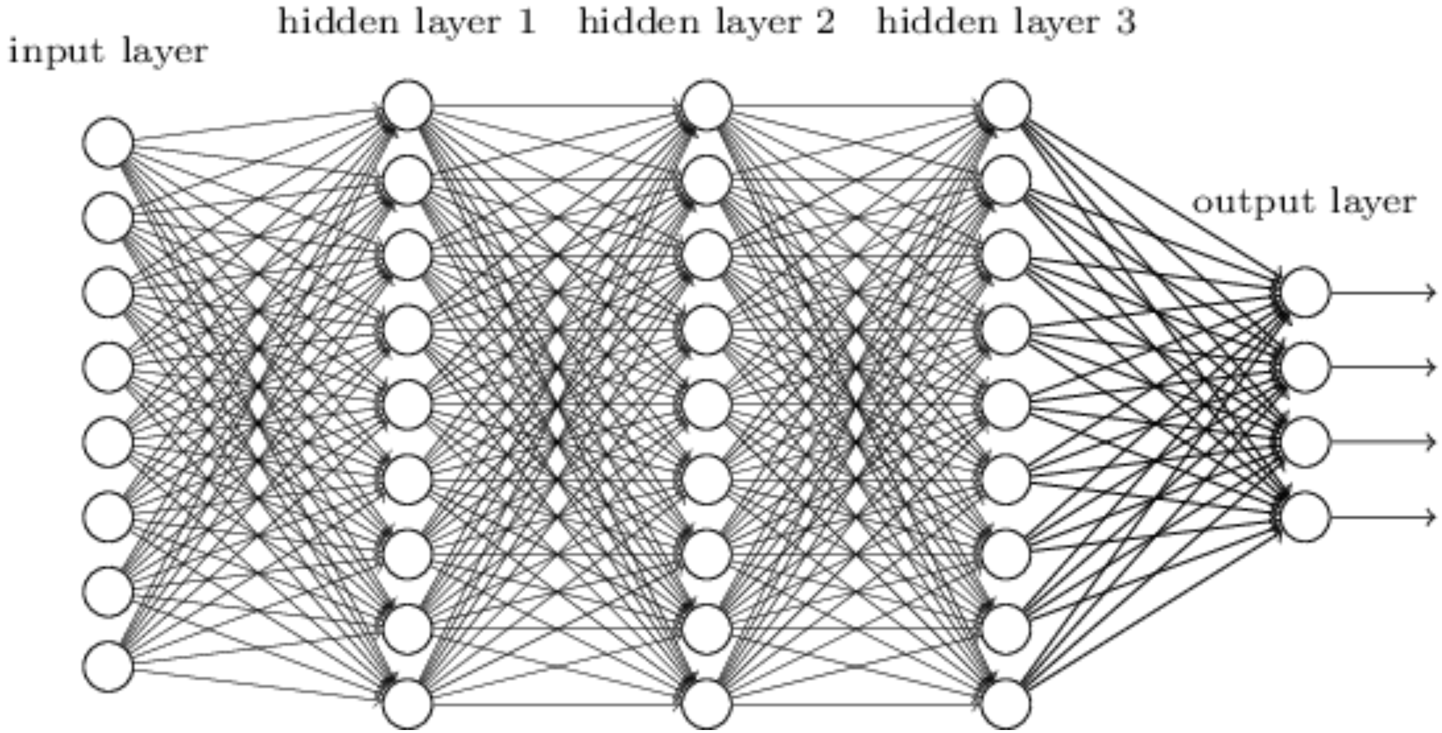
Image classification, detection, localization…

Feature learning: Going Deep



Deep learning

- Learn a *feature hierarchy* all the way from raw inputs (e.g. pixels) to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



Status Quo

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

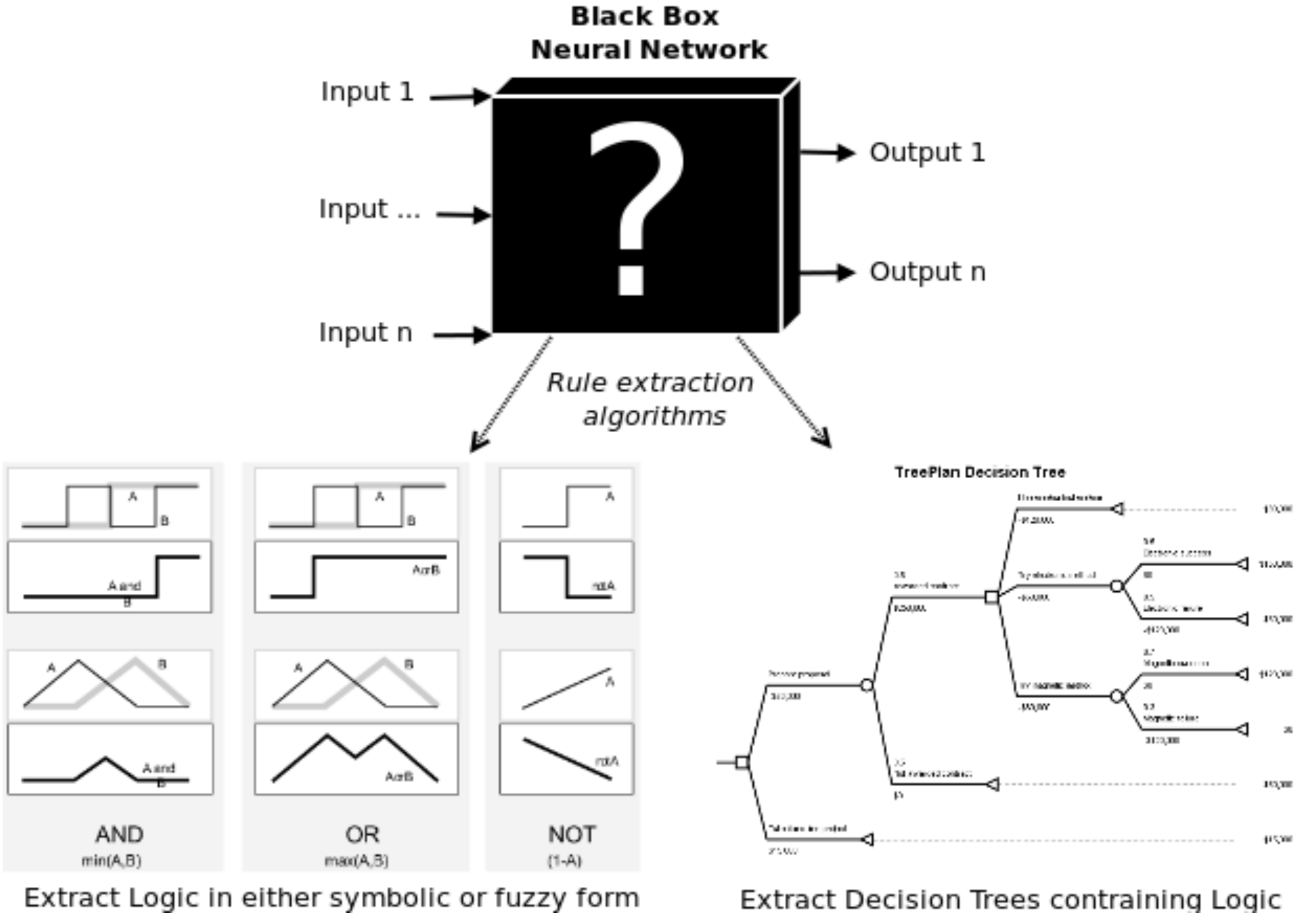
ResNet, 152 layers
(ILSVRC 2015)

Current Trend:

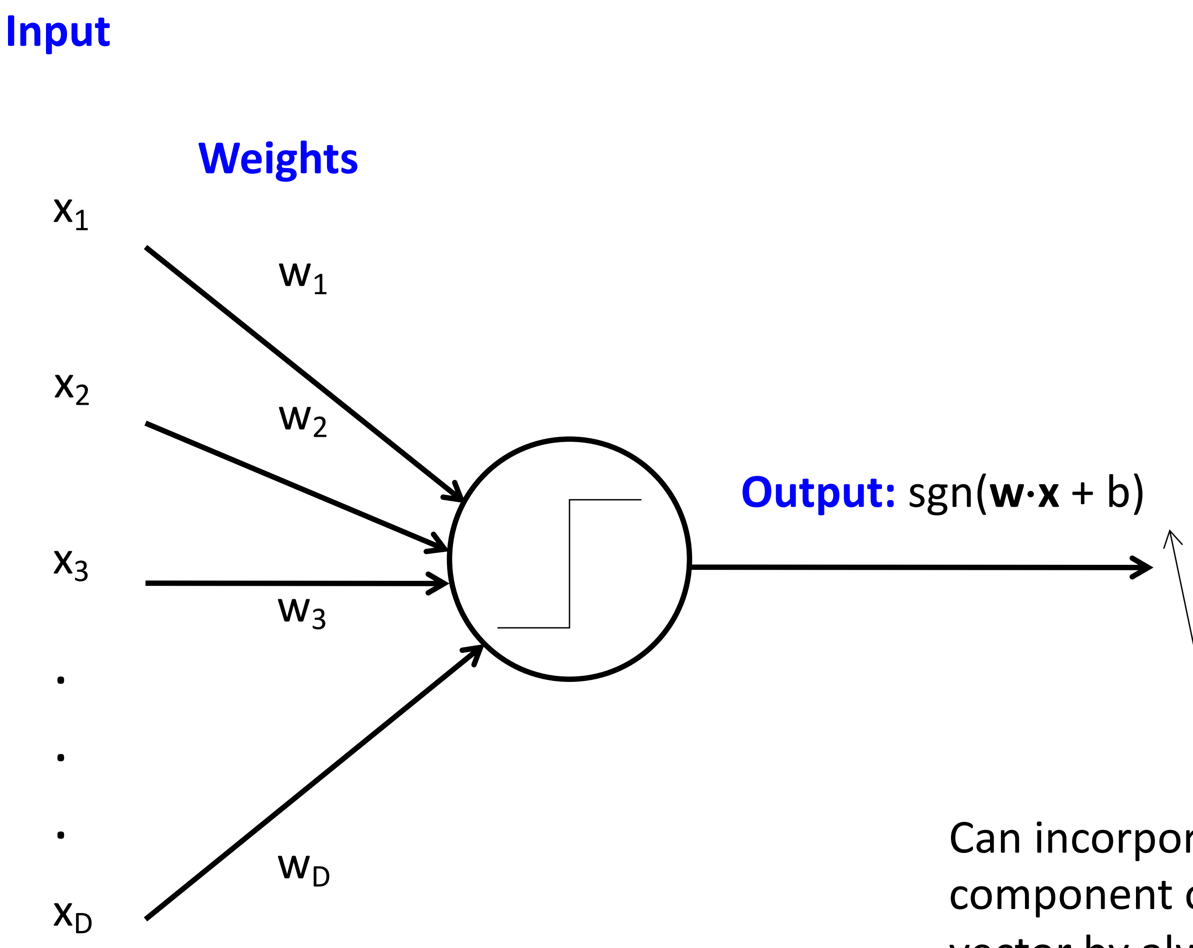
- To build increasingly larger, deeper networks, trained with more massive data, based on the benefits of high-performance computing.
- Play with the connectivity and add “skips”

Grand Challenges

- Why/how deep learning works?
  - *In theory, many cases shouldn't even work…*
  - Gap between engineering (or art) and science: Lack of theoretical understandings & guarantees, and analytical tools
  - Training is computationally expensive and difficult, relying on many "magics"
  - Lack of principled way to incorporate domain expertise, or to interpret the model behaviors

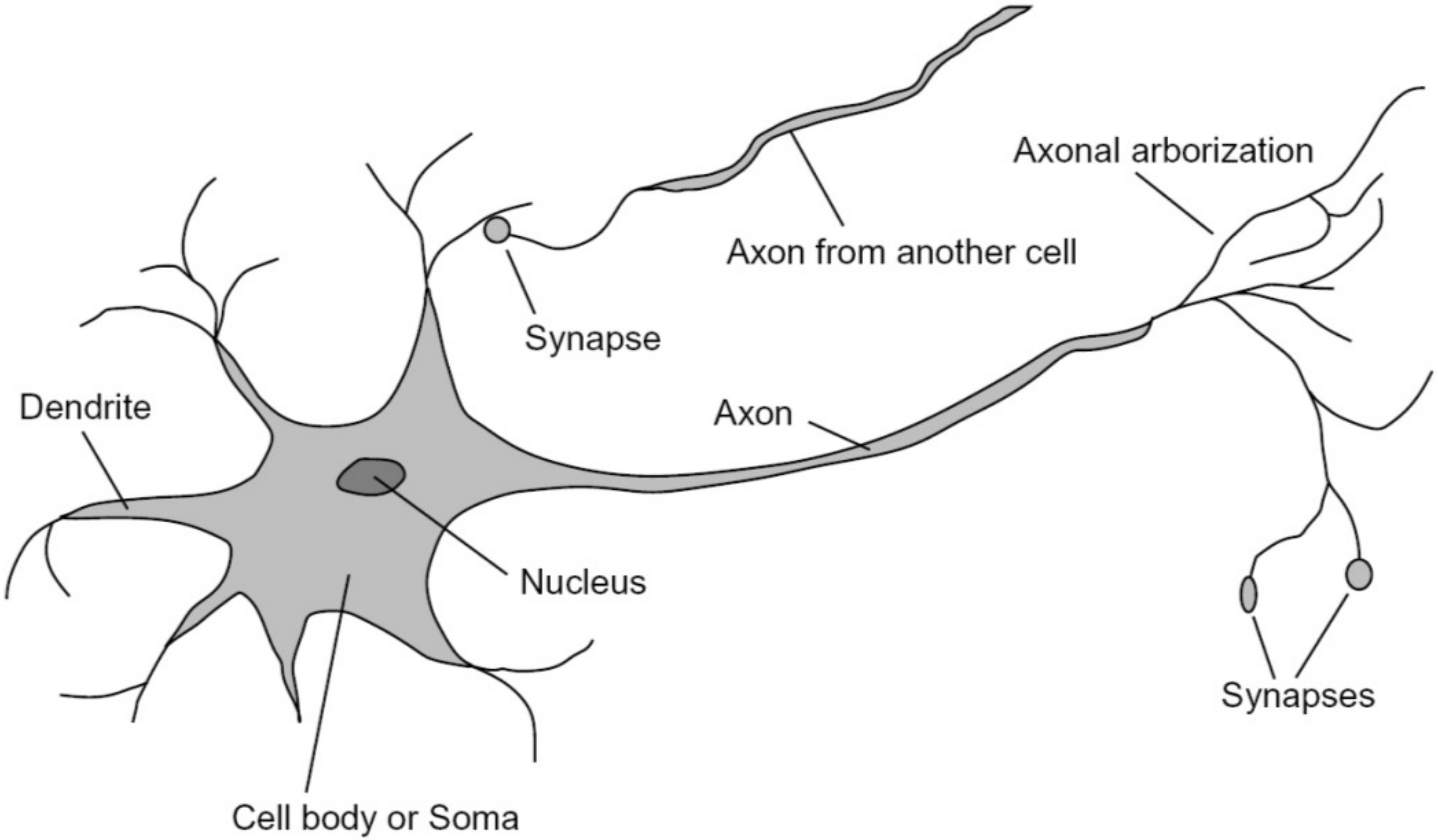


Perceptron



Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Loose inspiration: Human neurons



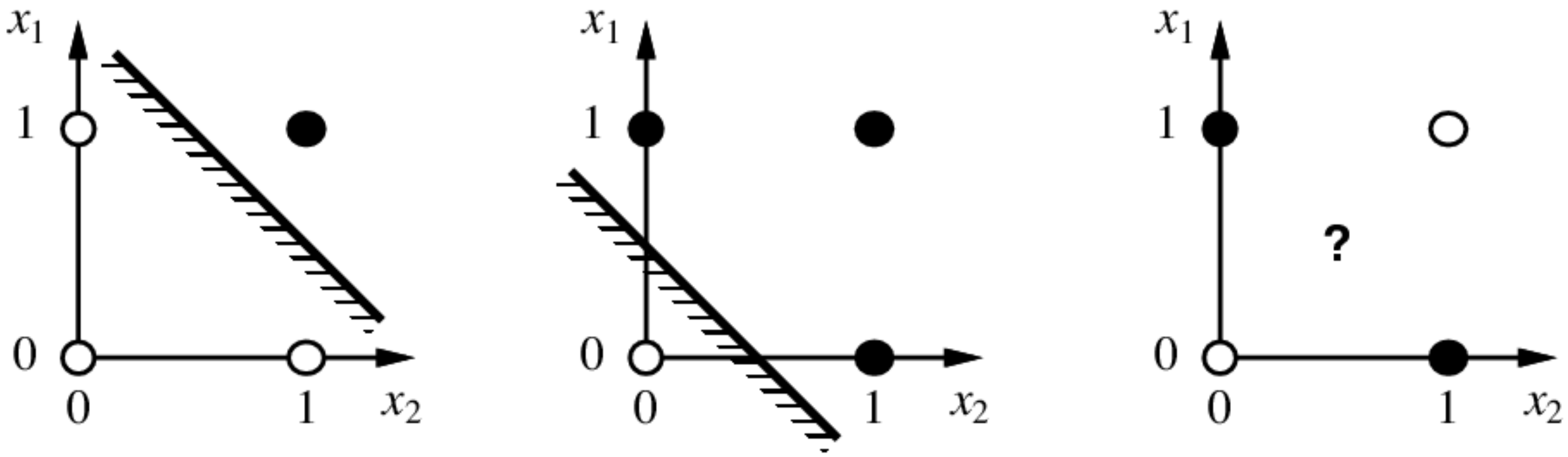
Perceptron training algorithm

- Initialize weights
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
  - Classify with current weights: $y' = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$
  - If classified incorrectly, update weights:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

  - α is a *learning rate* that should decay as a function of epoch t, e.g., 1000/(1000+t)

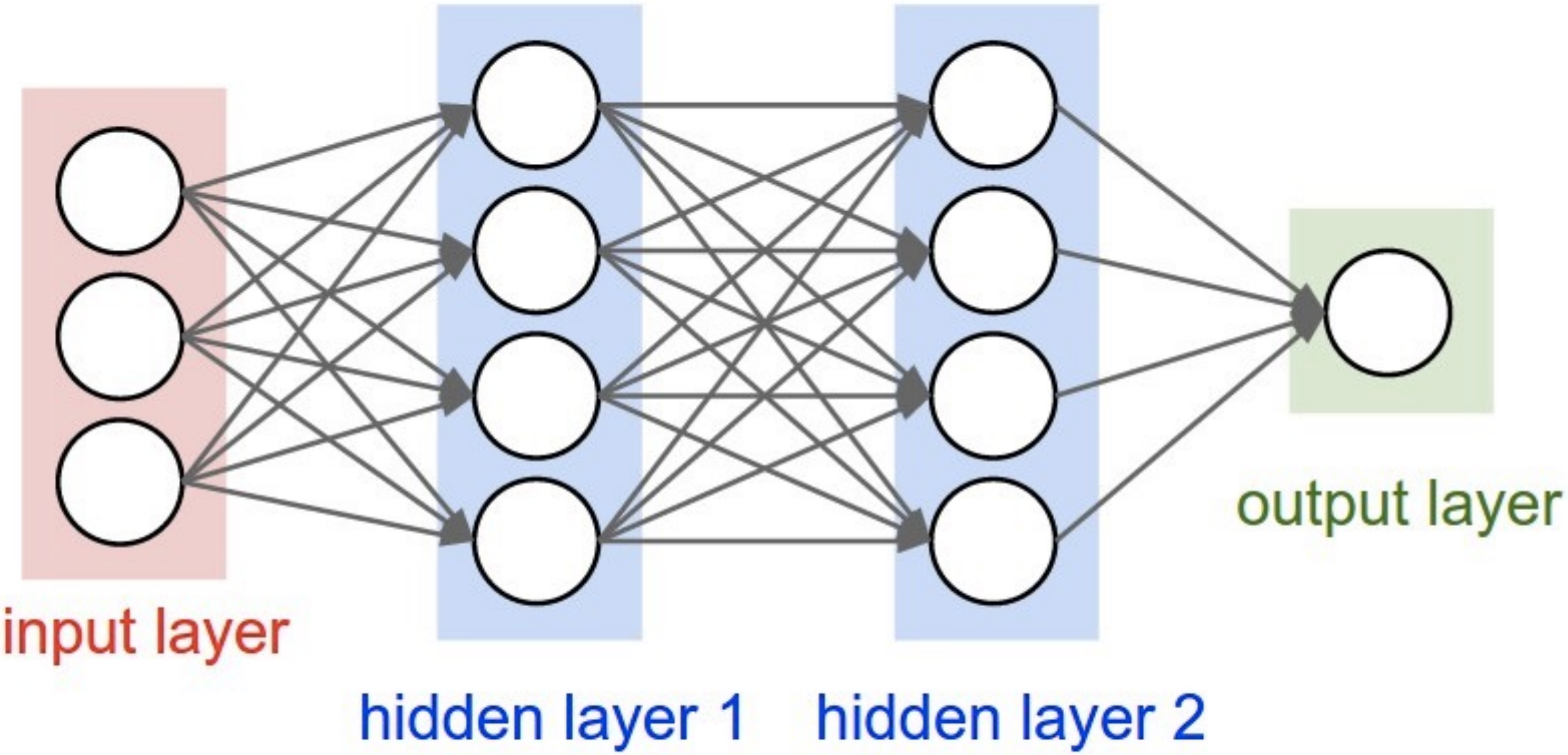
Linear separability



x_1 **and** x_2 x_1 **or** x_2 x_1 **xor** x_2

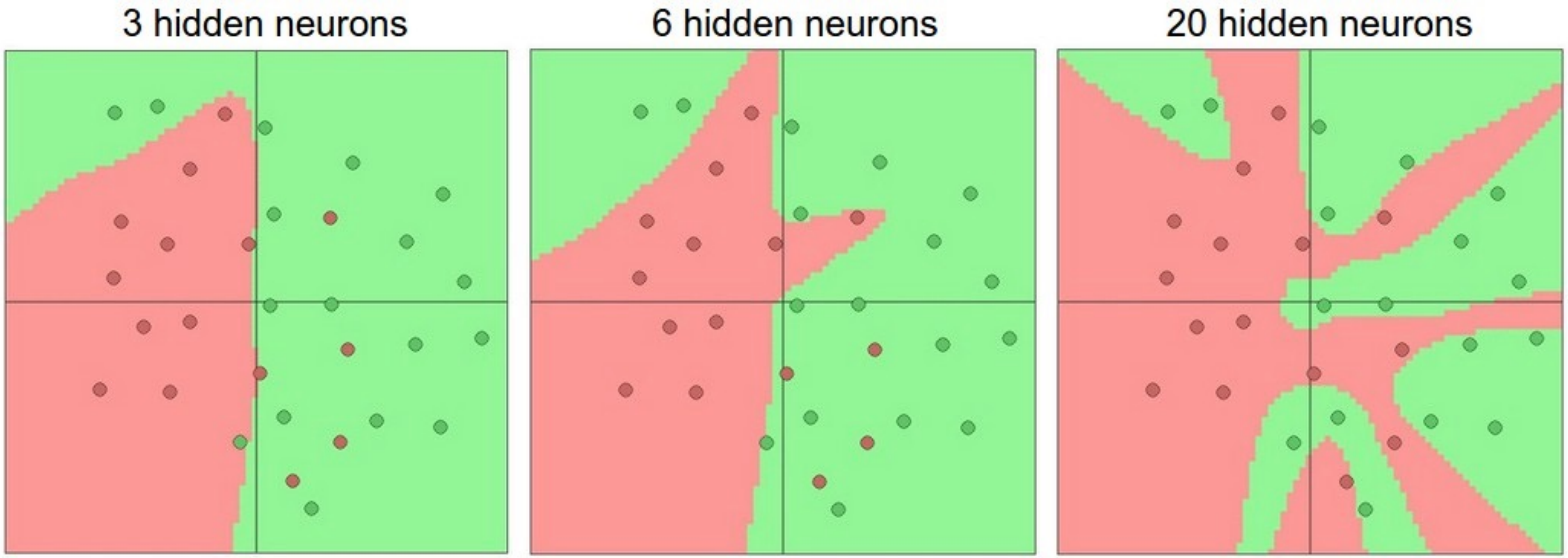
How do we make nonlinear classifiers out of perceptrons?

- Build a multi-layer neural network!



Network with a single hidden layer

- Hidden layer size and *network capacity*:



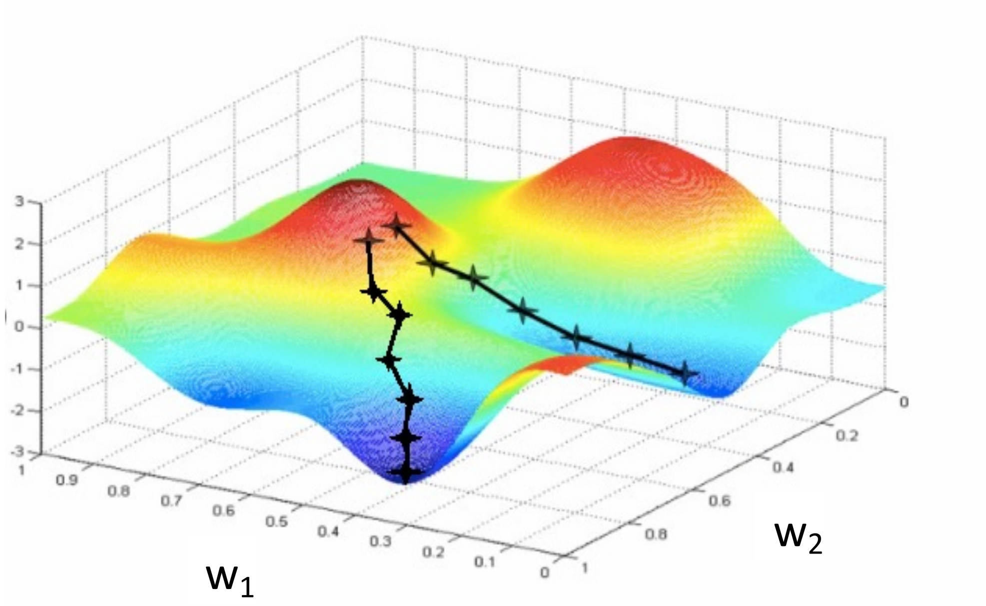
Source: http://cs231n.github.io/neural-networks-1/

Training of multi-layer networks

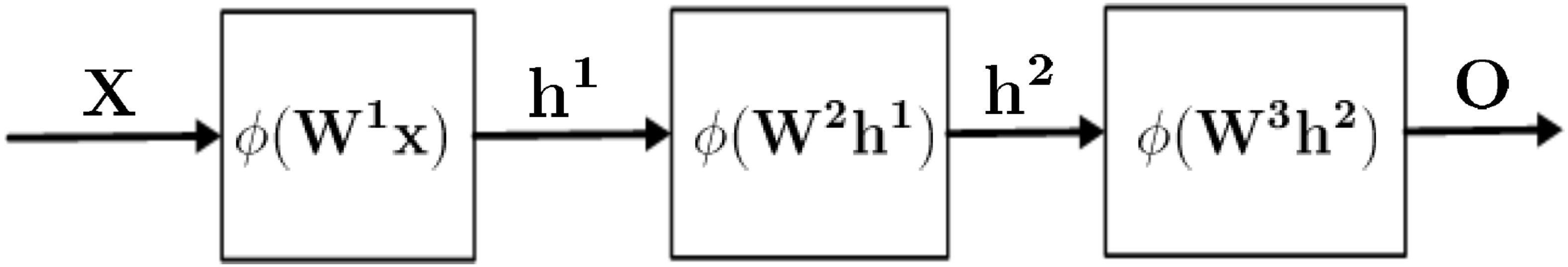
- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

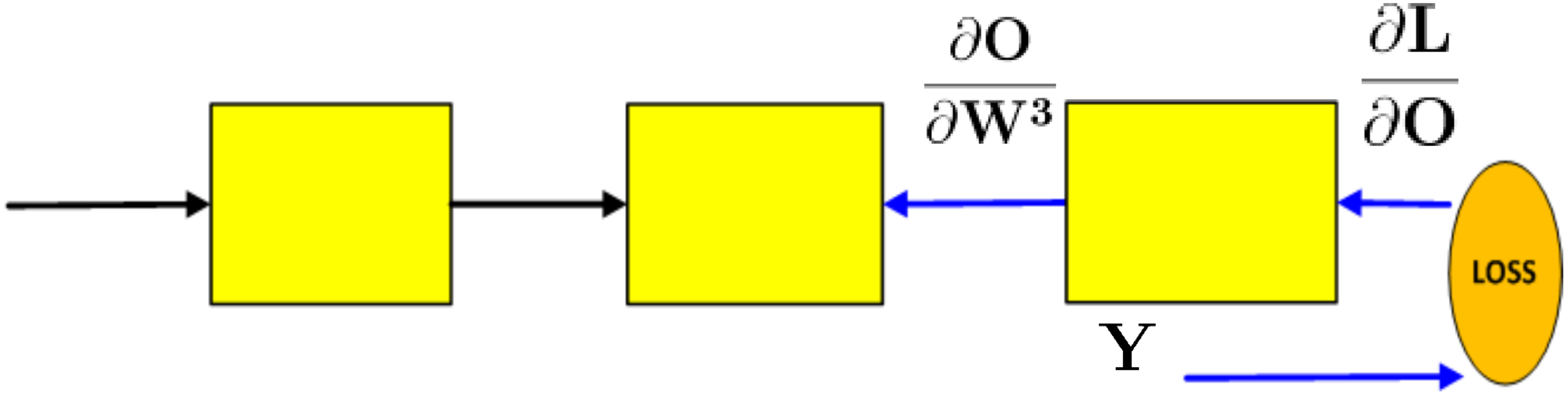
- Update weights by **gradient descent:** $$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$



Forward-Backward Propagation



Forward propagation: $h(\mathbf{x}) = \phi(\mathbf{Wx})$

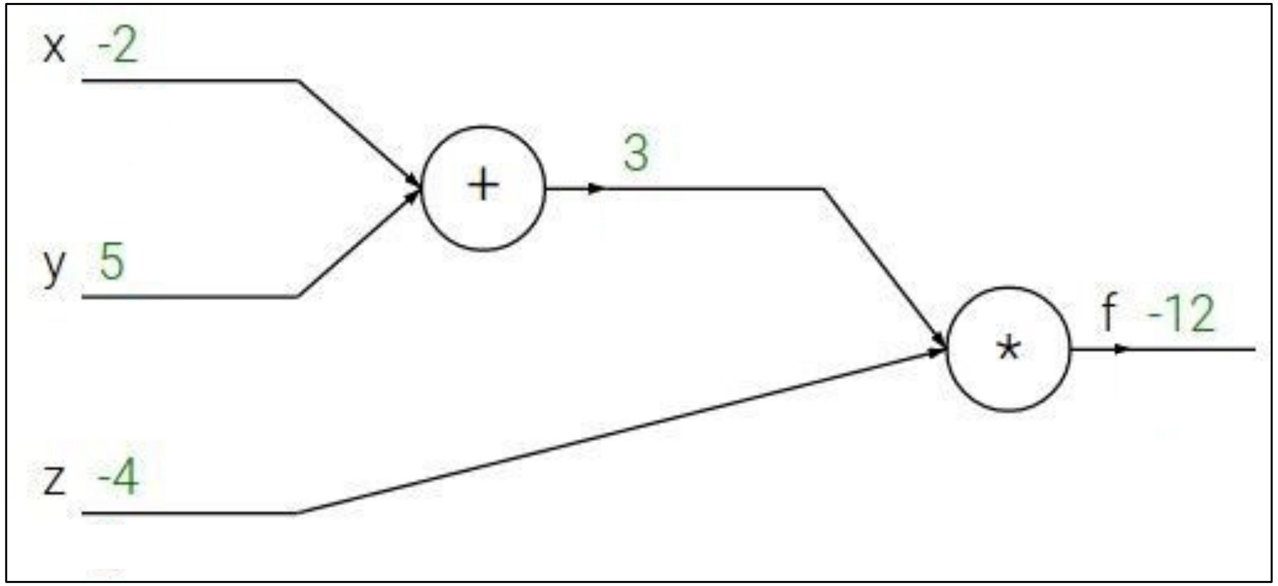


Backward propagation: $\frac{\partial \mathbf{L}}{\partial \mathbf{W^3}} = \frac{\partial \mathbf{L}}{\partial \mathbf{O}} \frac{\partial \mathbf{O}}{\partial \mathbf{W^3}}$ **(Chain Rule)**

Backpropagation: a simple example

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

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



Backpropagation: a simple example

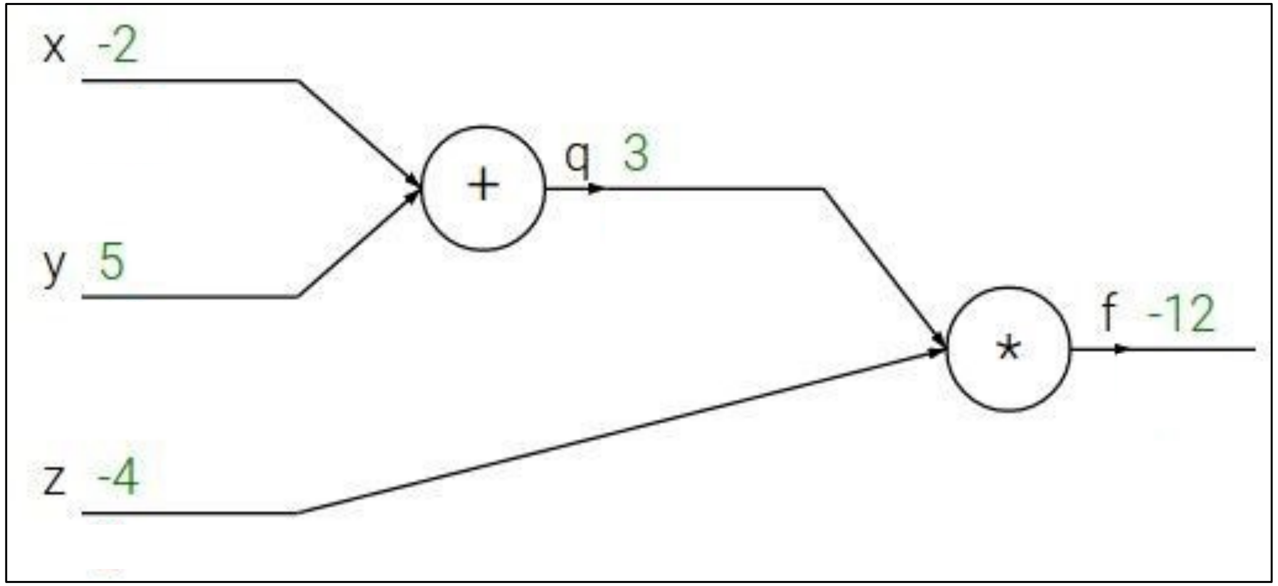
$$f(x, y, z) = (x + y)z$$

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



Backpropagation: a simple example

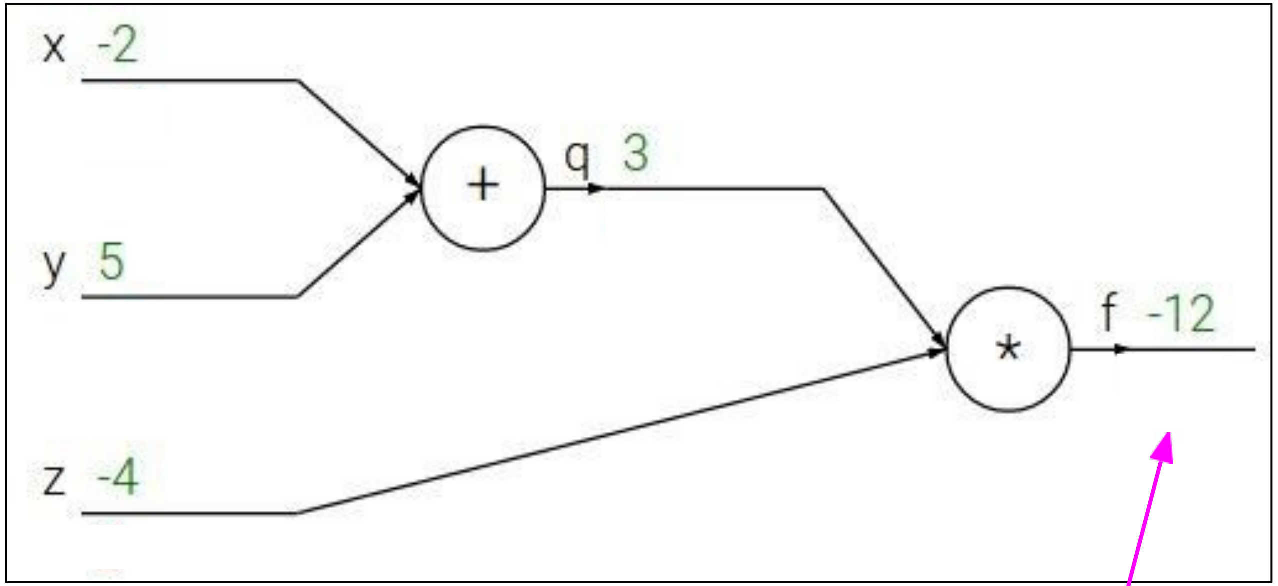
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

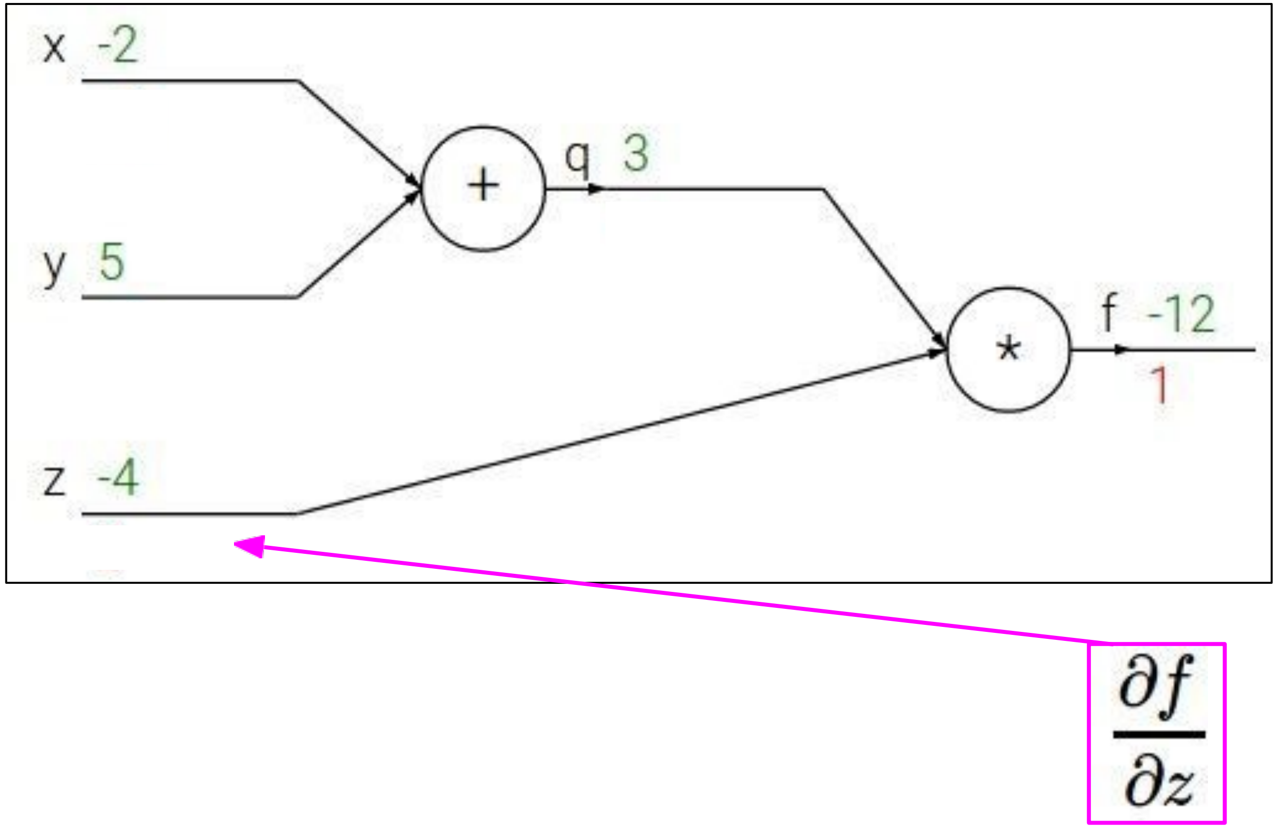
$$f(x, y, z) = (x + y)z$$

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



Backpropagation: a simple example

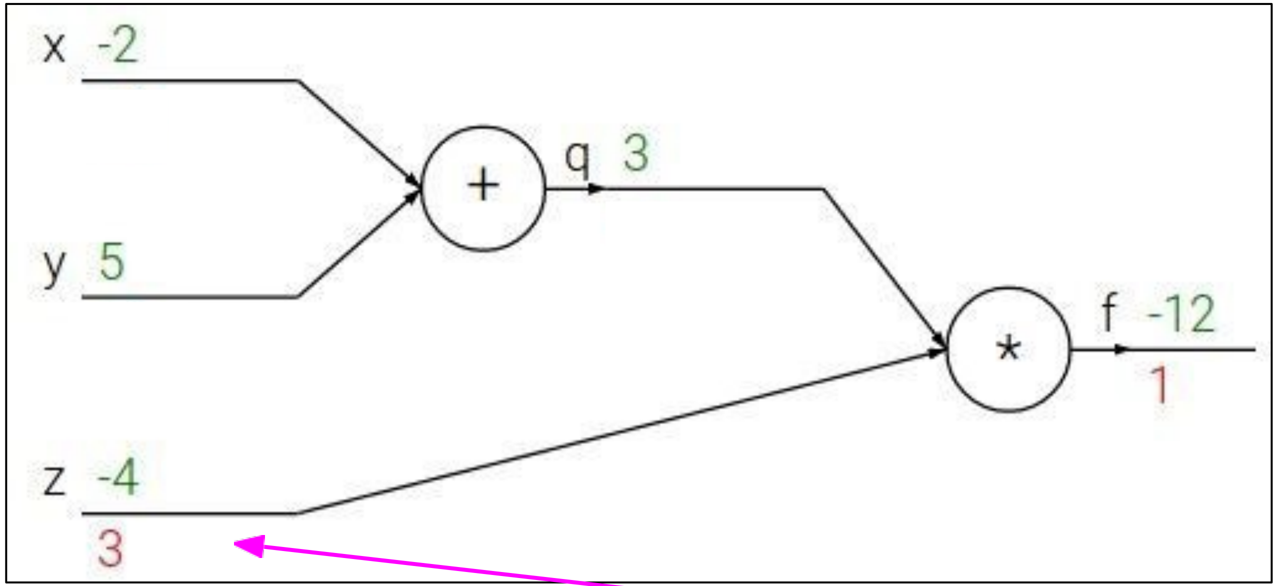
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

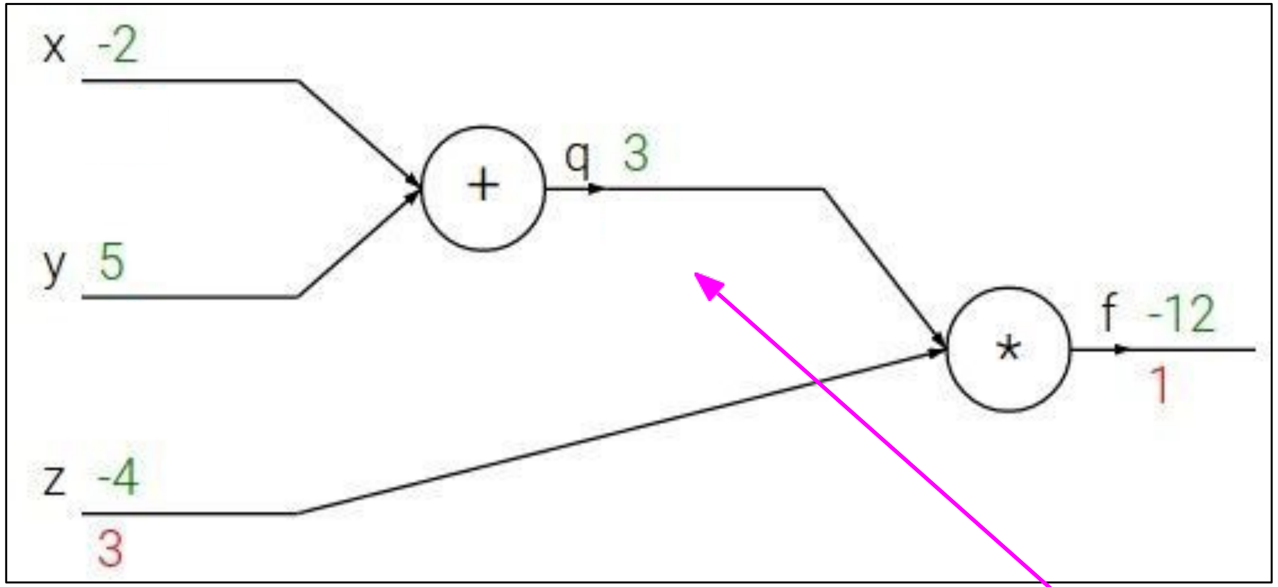
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

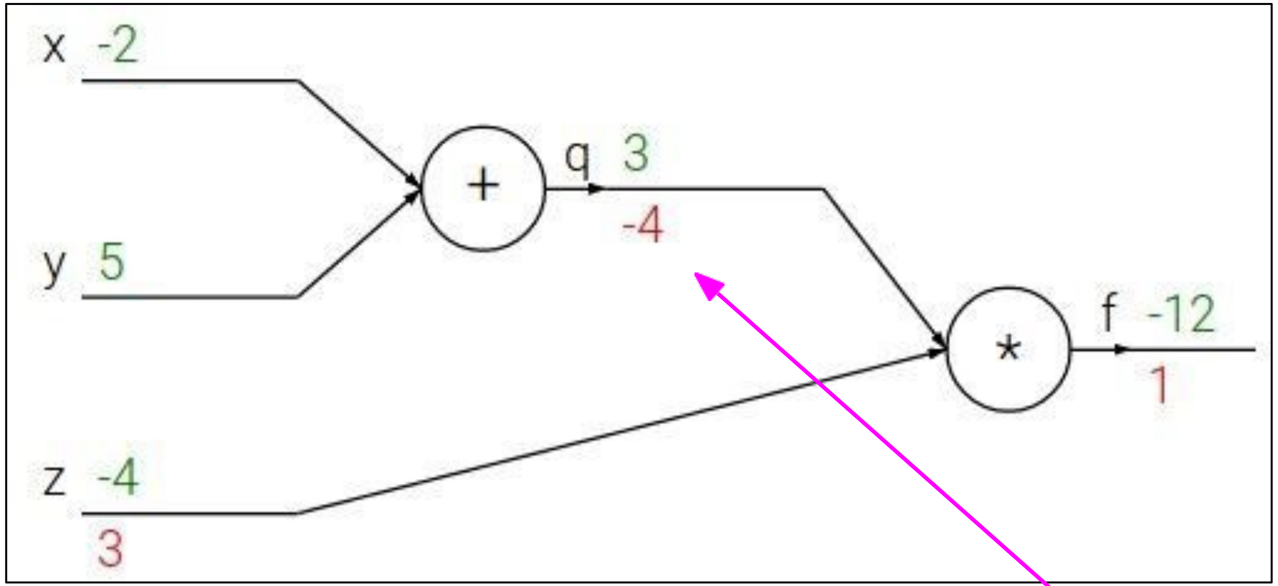
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

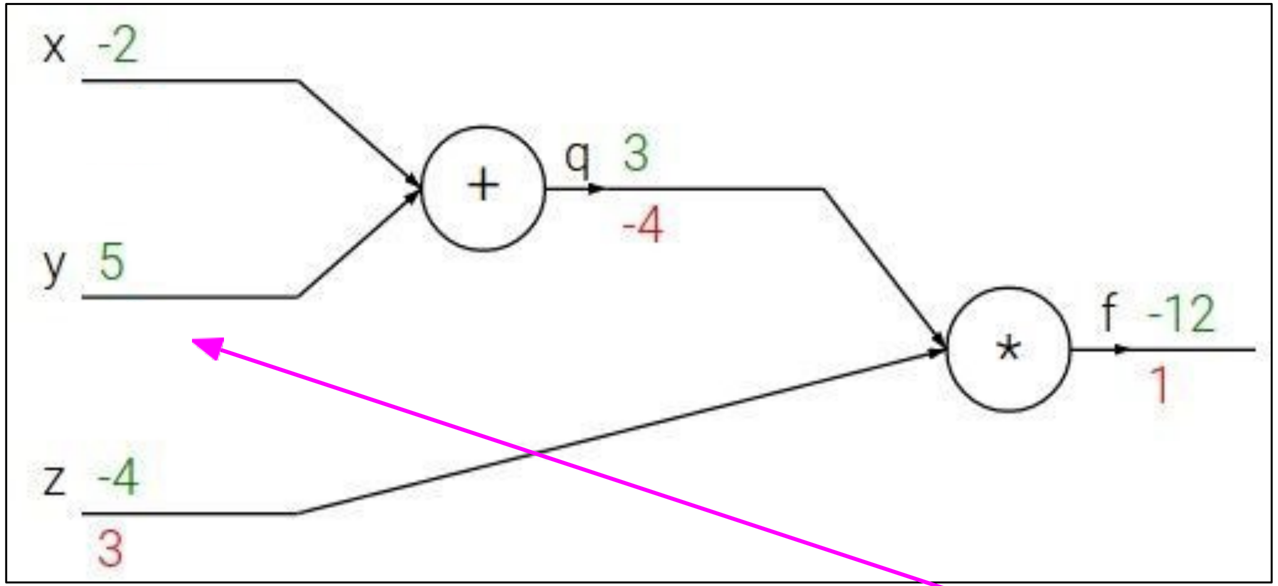
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

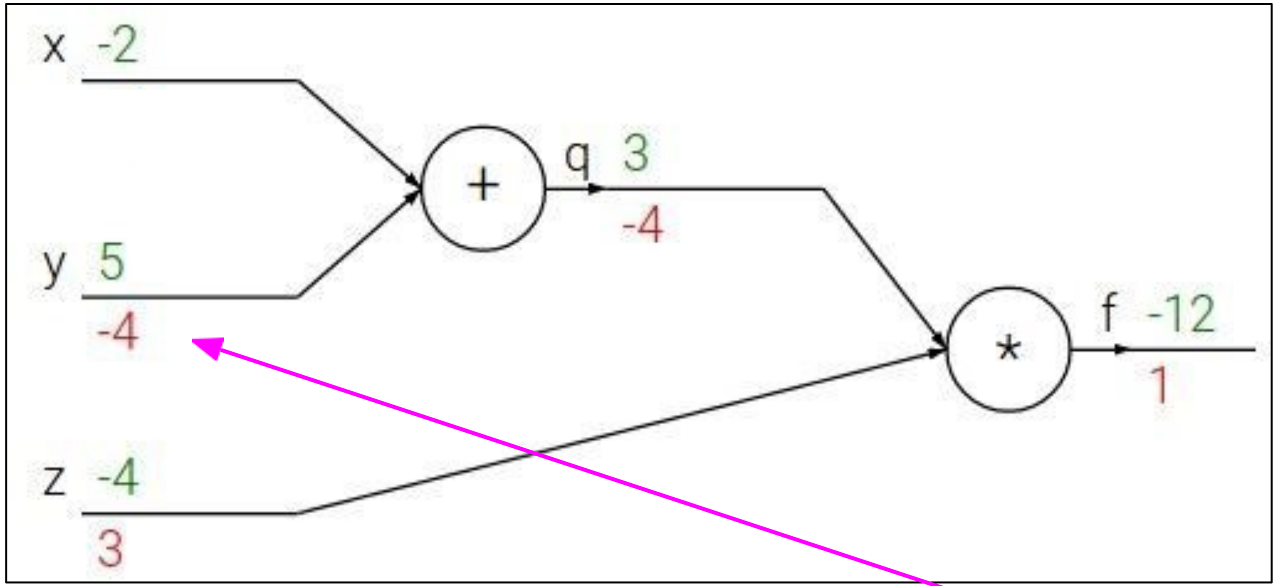
$$f(x, y, z) = (x + y)z$$

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



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

Chain rule:

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

Backpropagation: a simple example

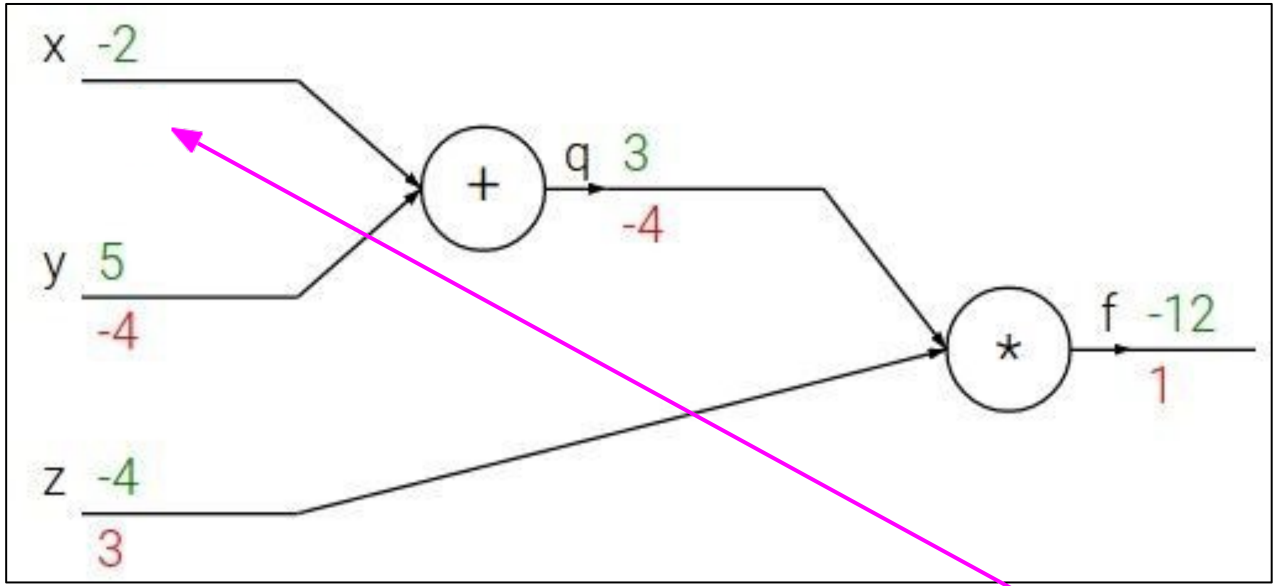
$$f(x, y, z) = (x + y)z$$

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



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

Backpropagation: a simple example

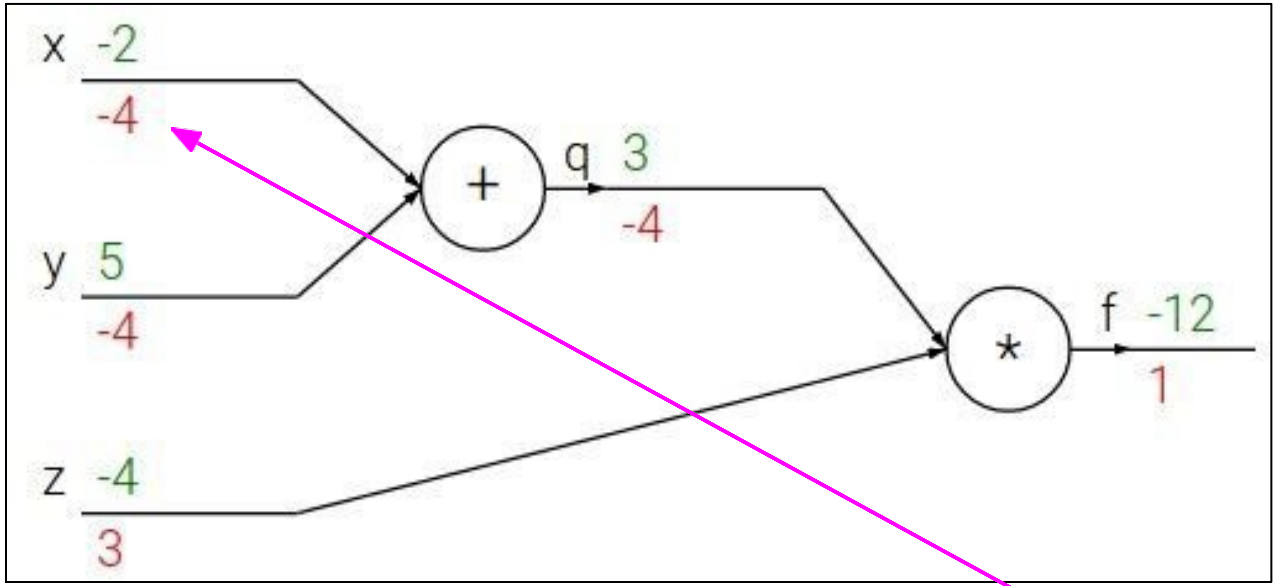
$$f(x, y, z) = (x + y)z$$

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



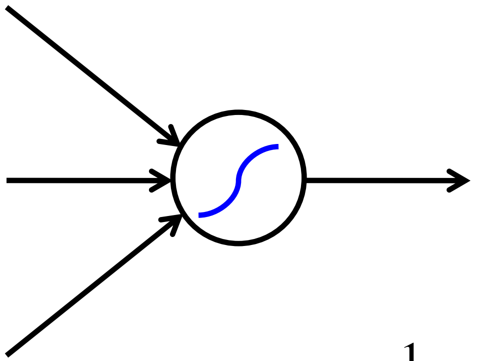
$$\frac{\partial f}{\partial x}$$

Chain rule:

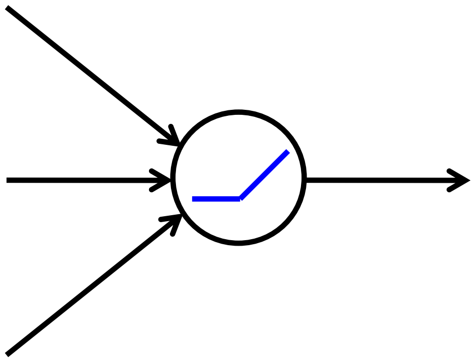
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Training of multi-layer nonlinear networks

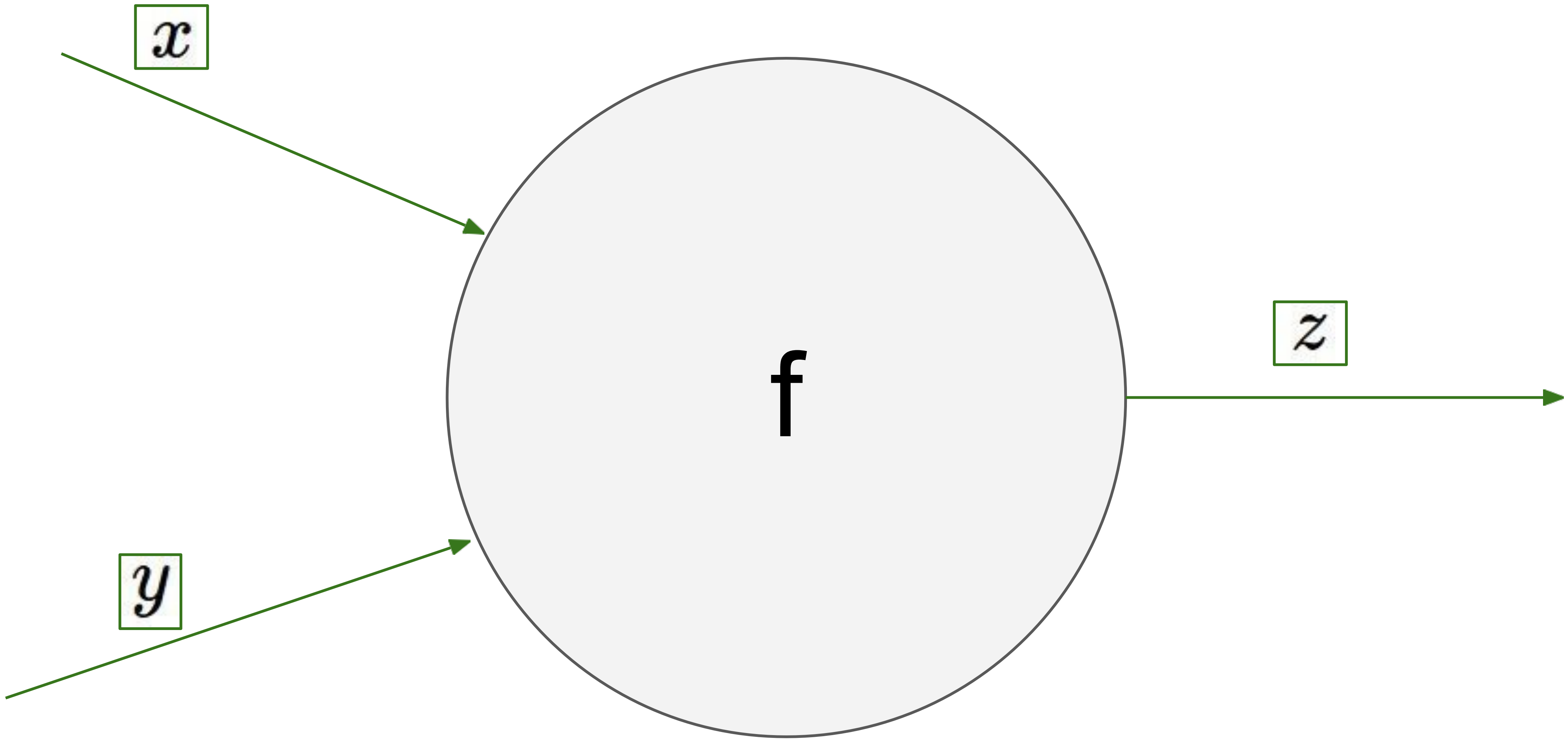
- **Gradient descent** requires neural networks to be equipped with a (nearly) differentiable nonlinearity function, called neuron

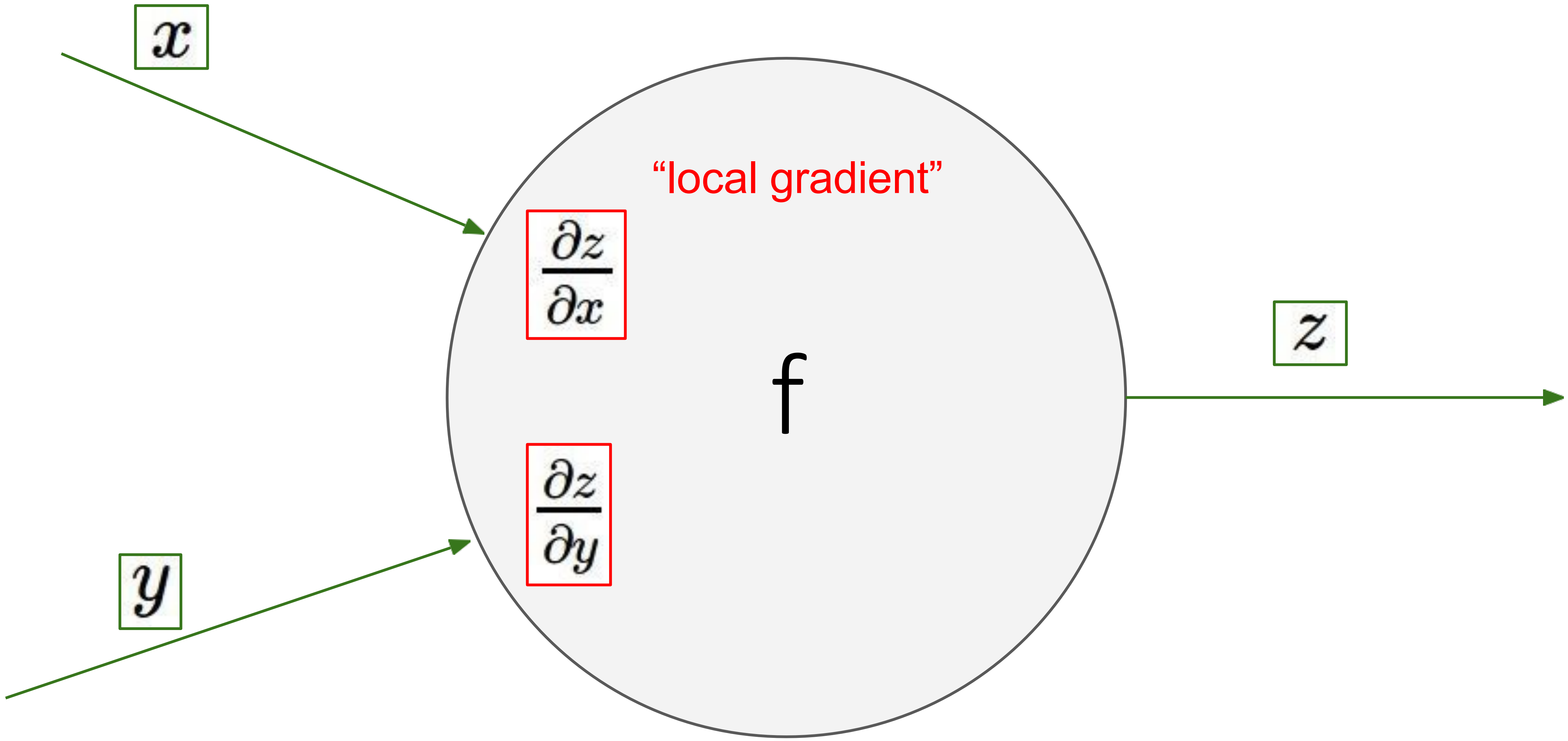


Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$

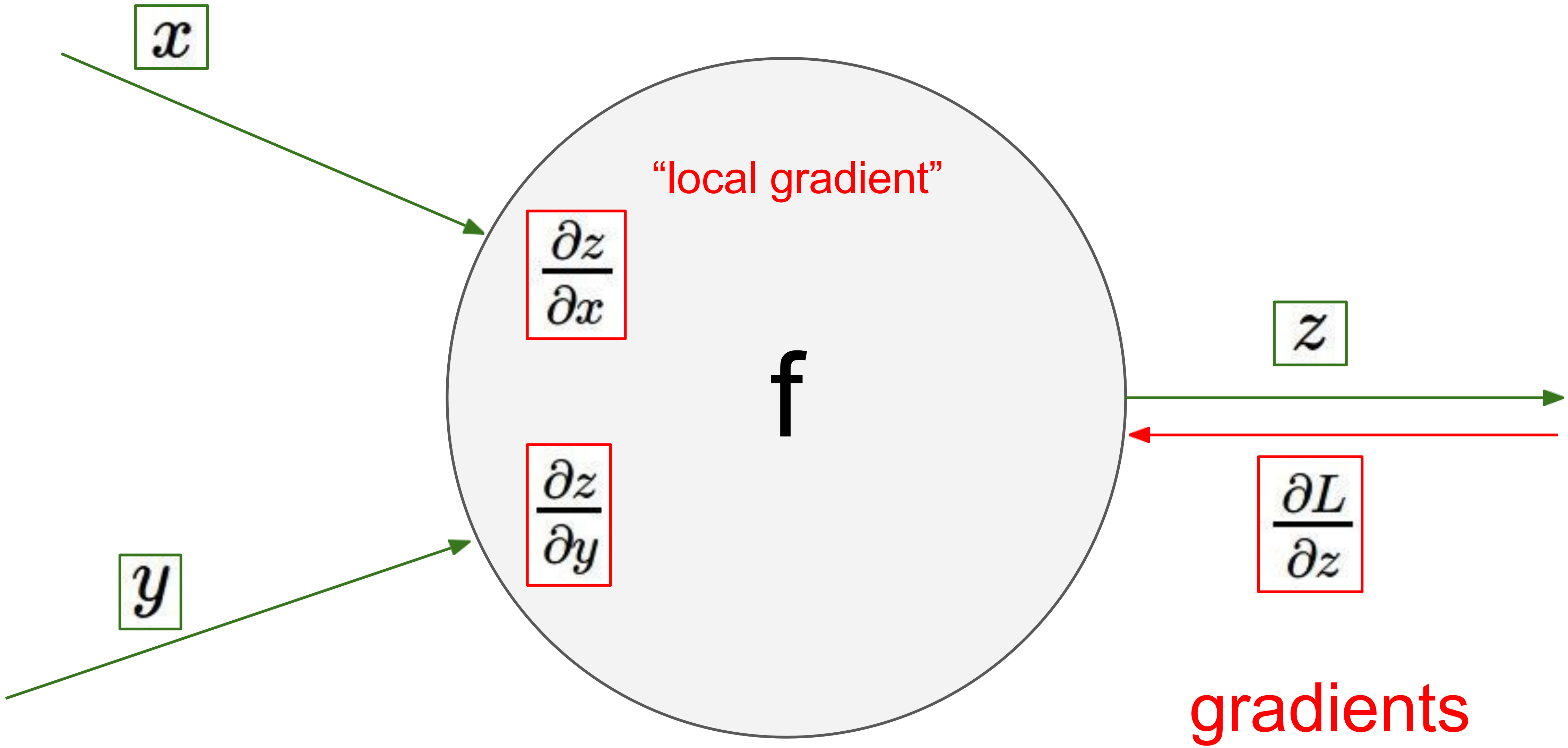


Rectified linear unit (ReLU): $g(t) = \max(0,t)$

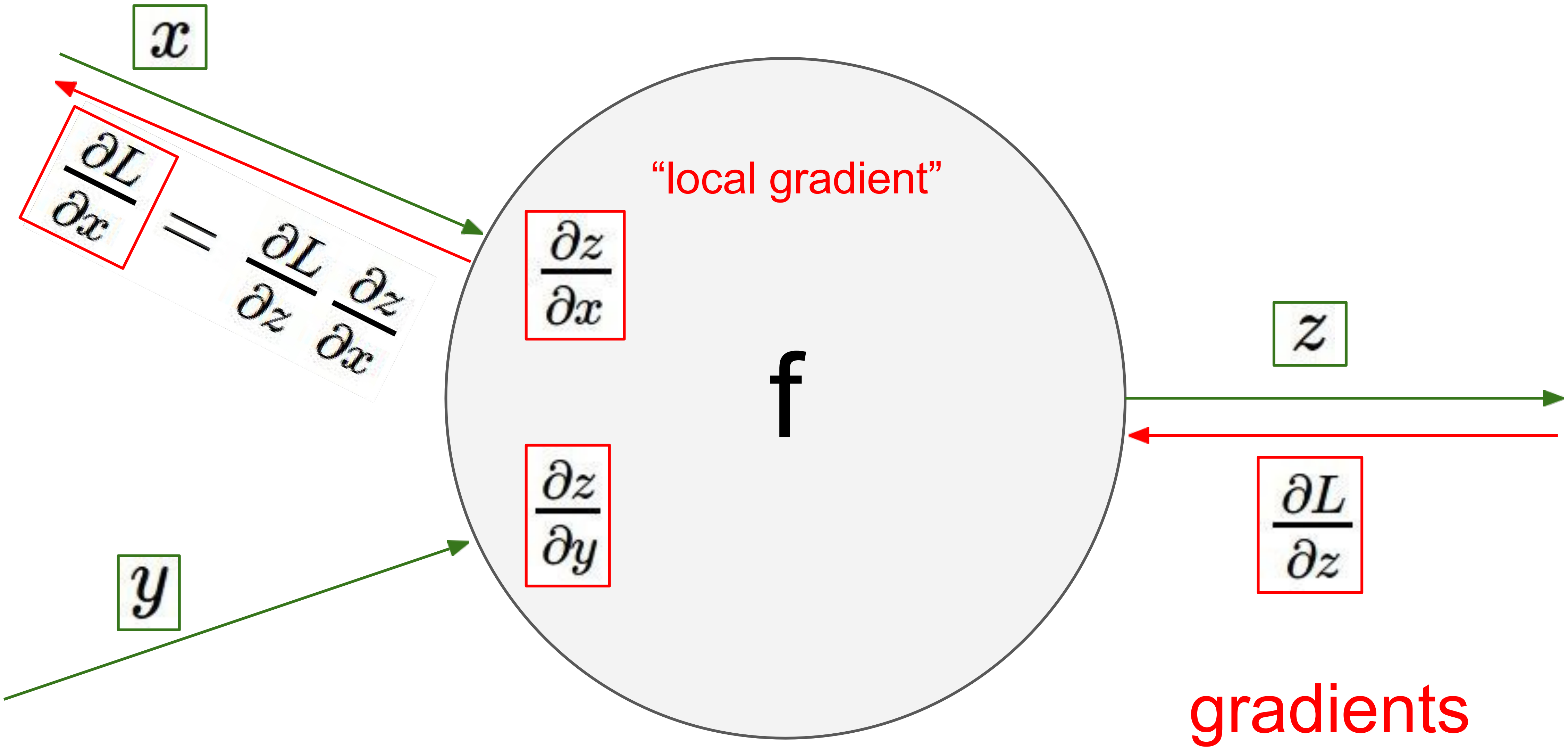

x
y
f
z



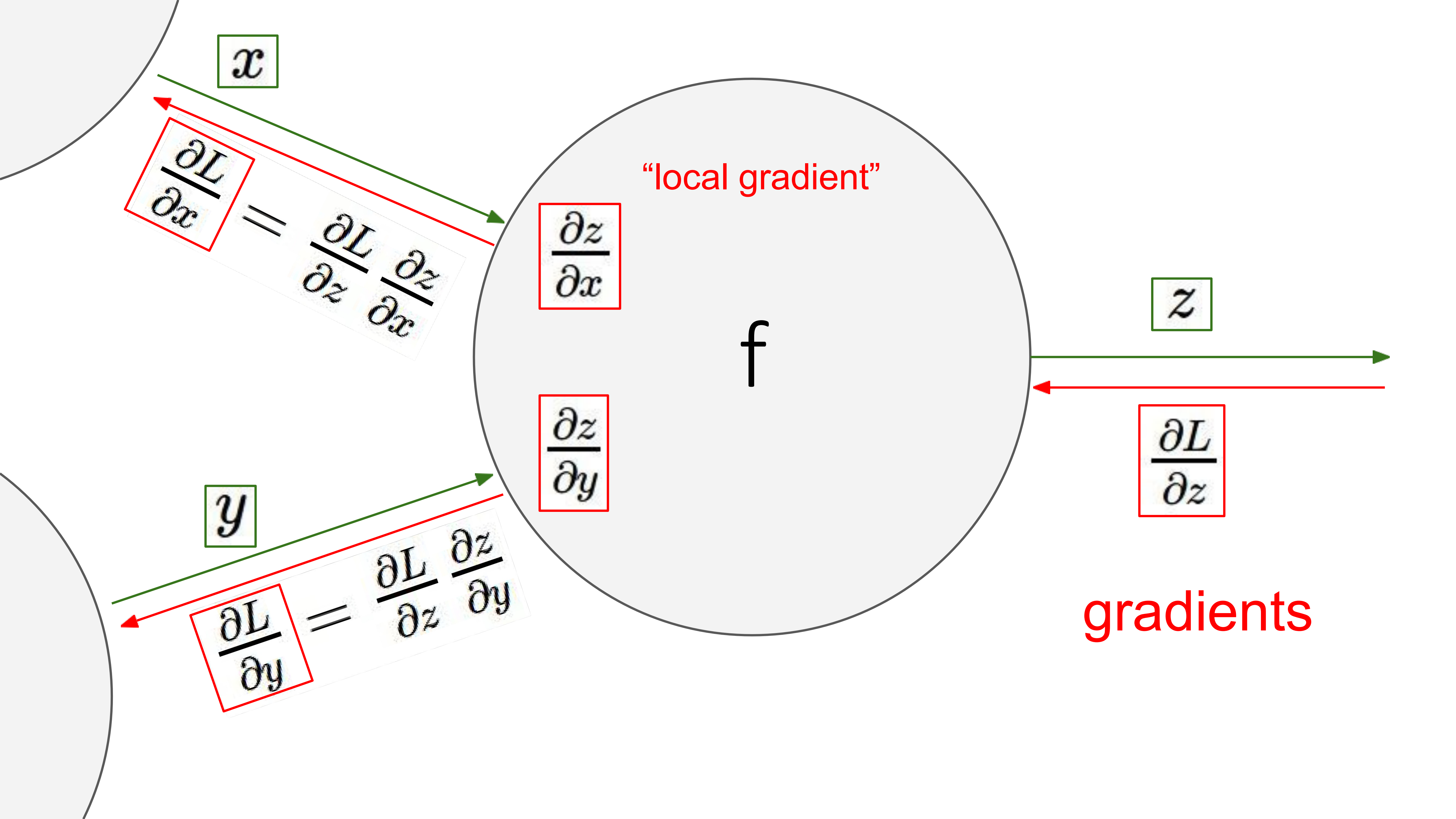
x
y
z
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$



x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
gradients

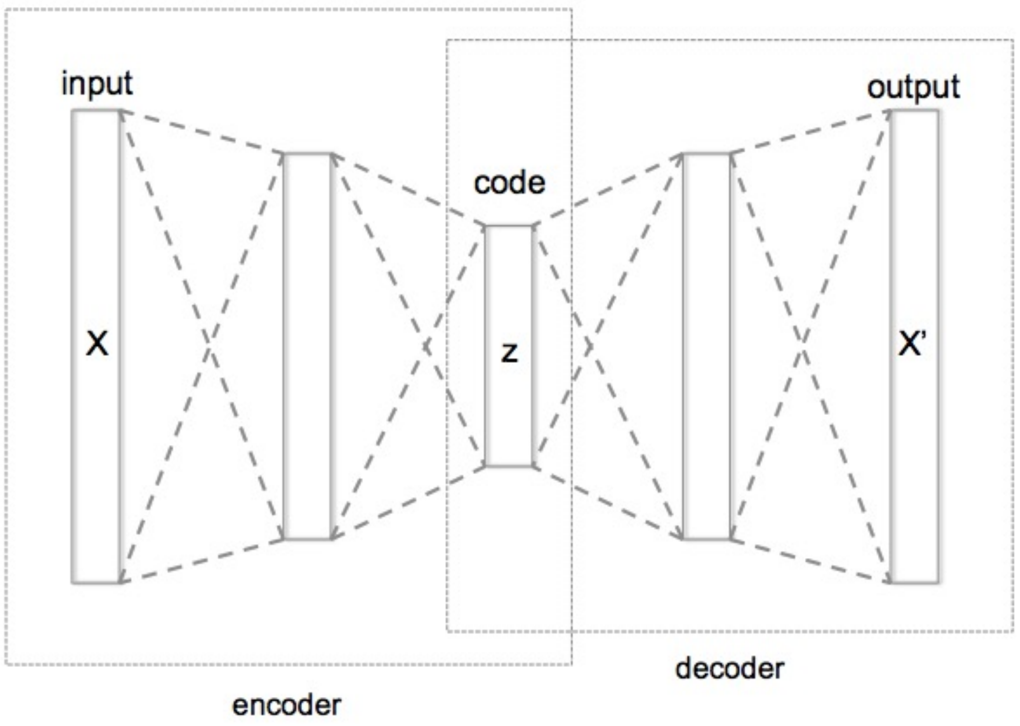


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
y
z
$\frac{\partial L}{\partial z}$
gradients


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
“local gradient”
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Auto-Encoder

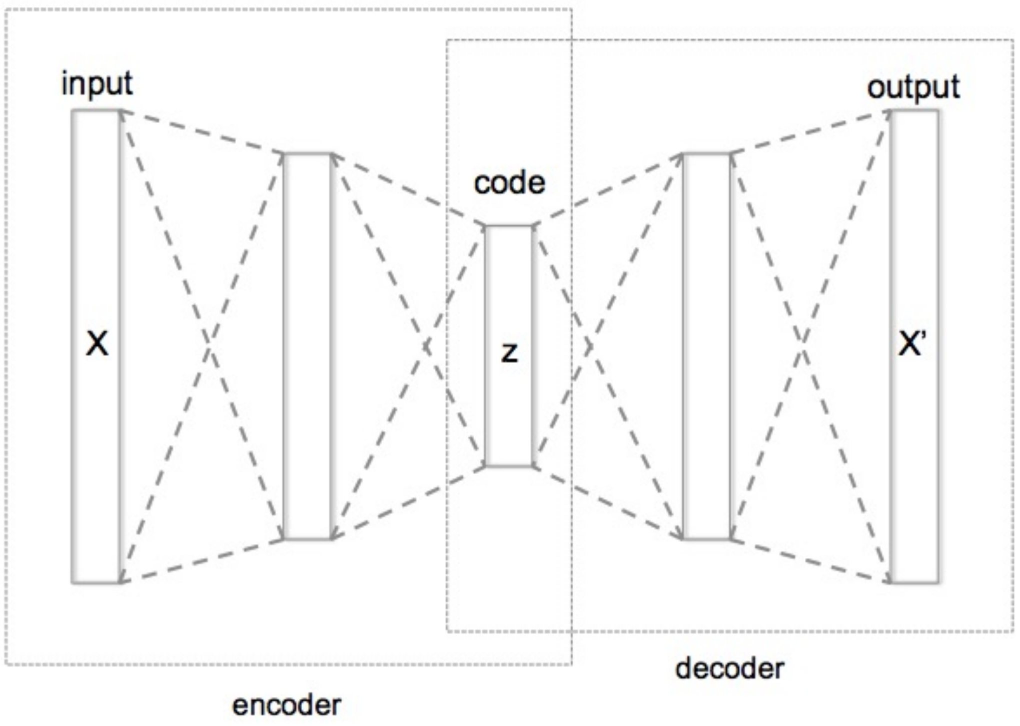
- Unsupervised feature extraction
- Reconstruct the input from itself via using “bottleneck”



X = X’

Denoising Auto-Encoder

- Reconstruct the input from a slightly corrupted “noisy” version
- Purpose: learning robust features for better generalization

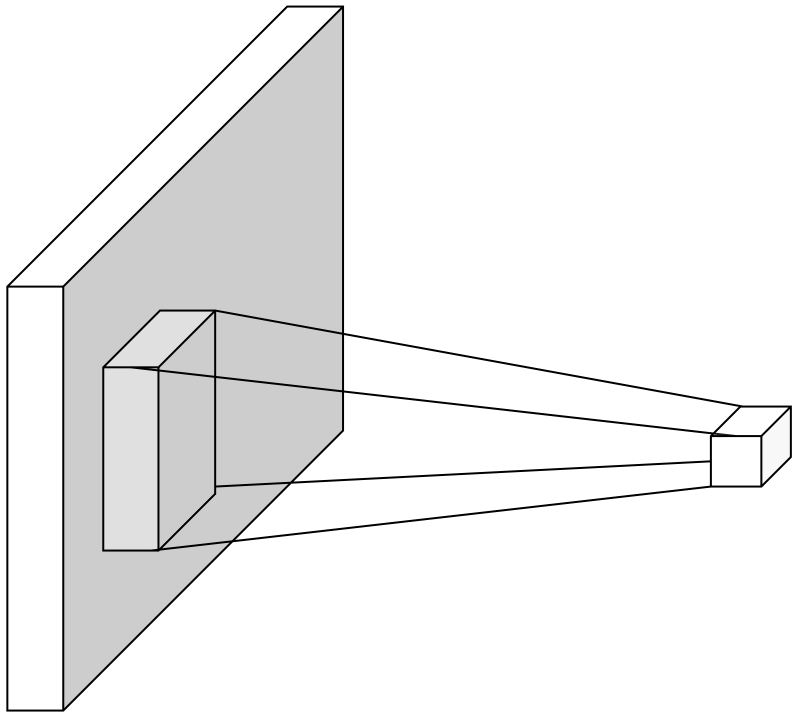


$X = X' + noise$

From NNs to Convolution NNs

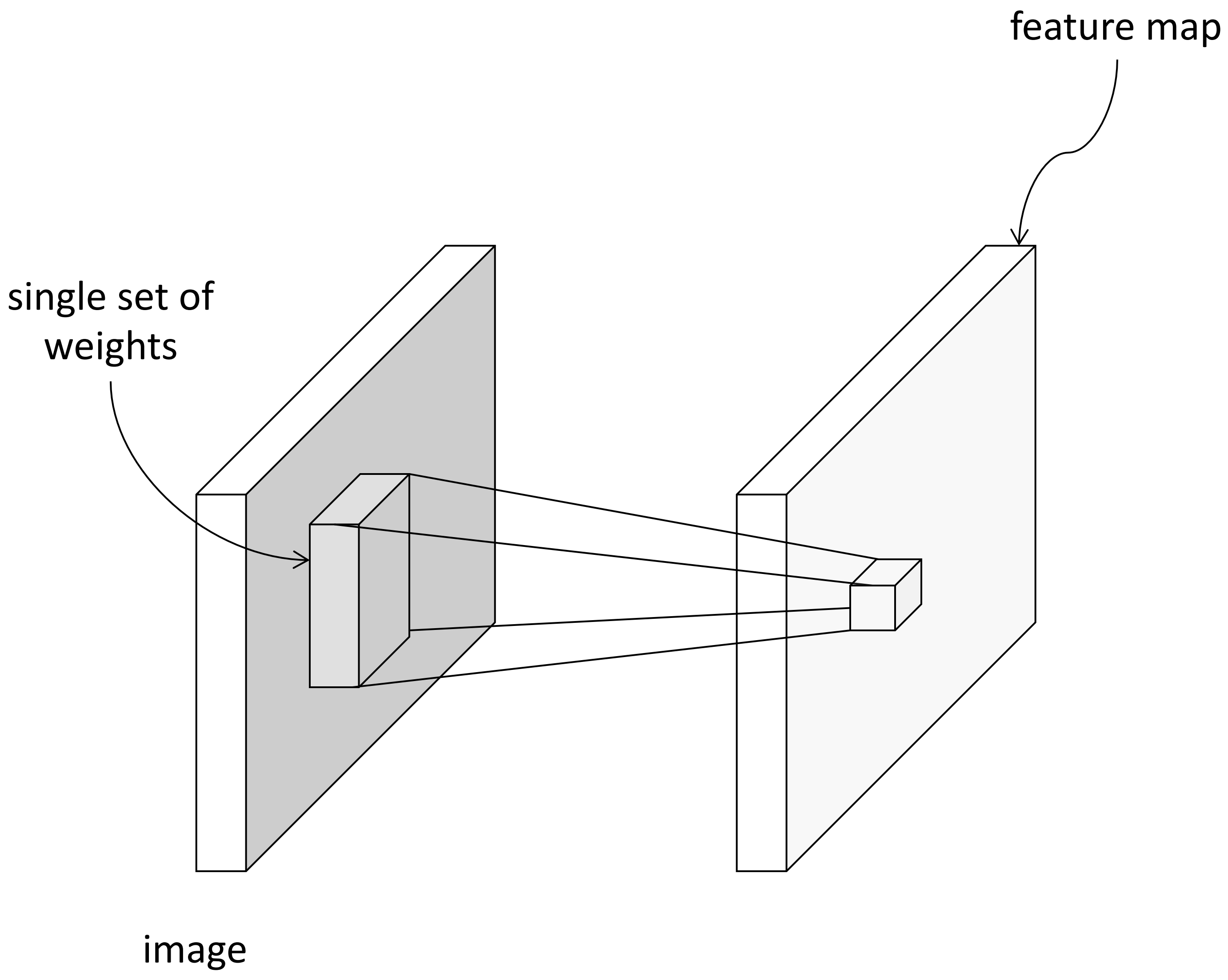
The most important building block in modern deep learning

From fully connected to convolutional networks

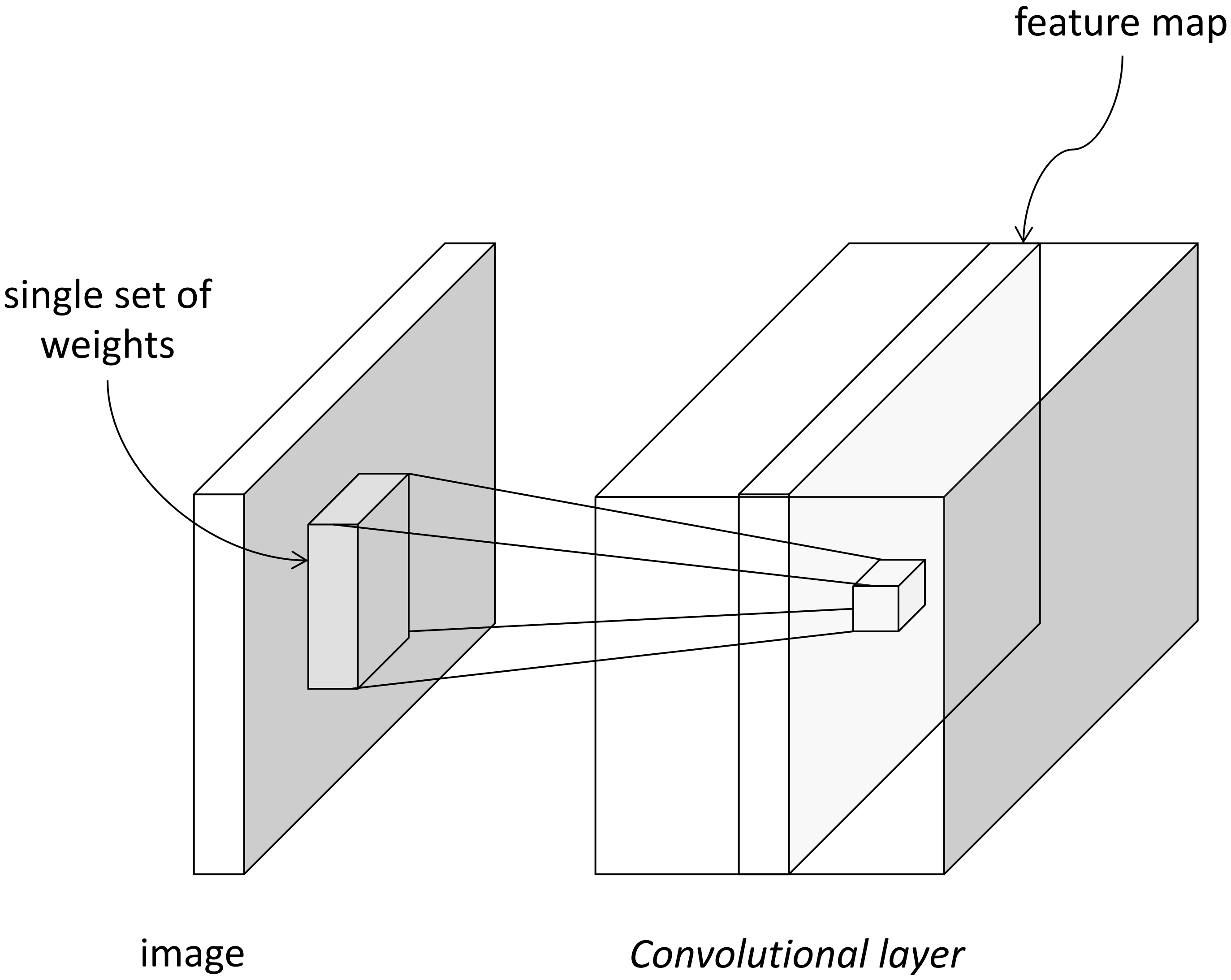


image

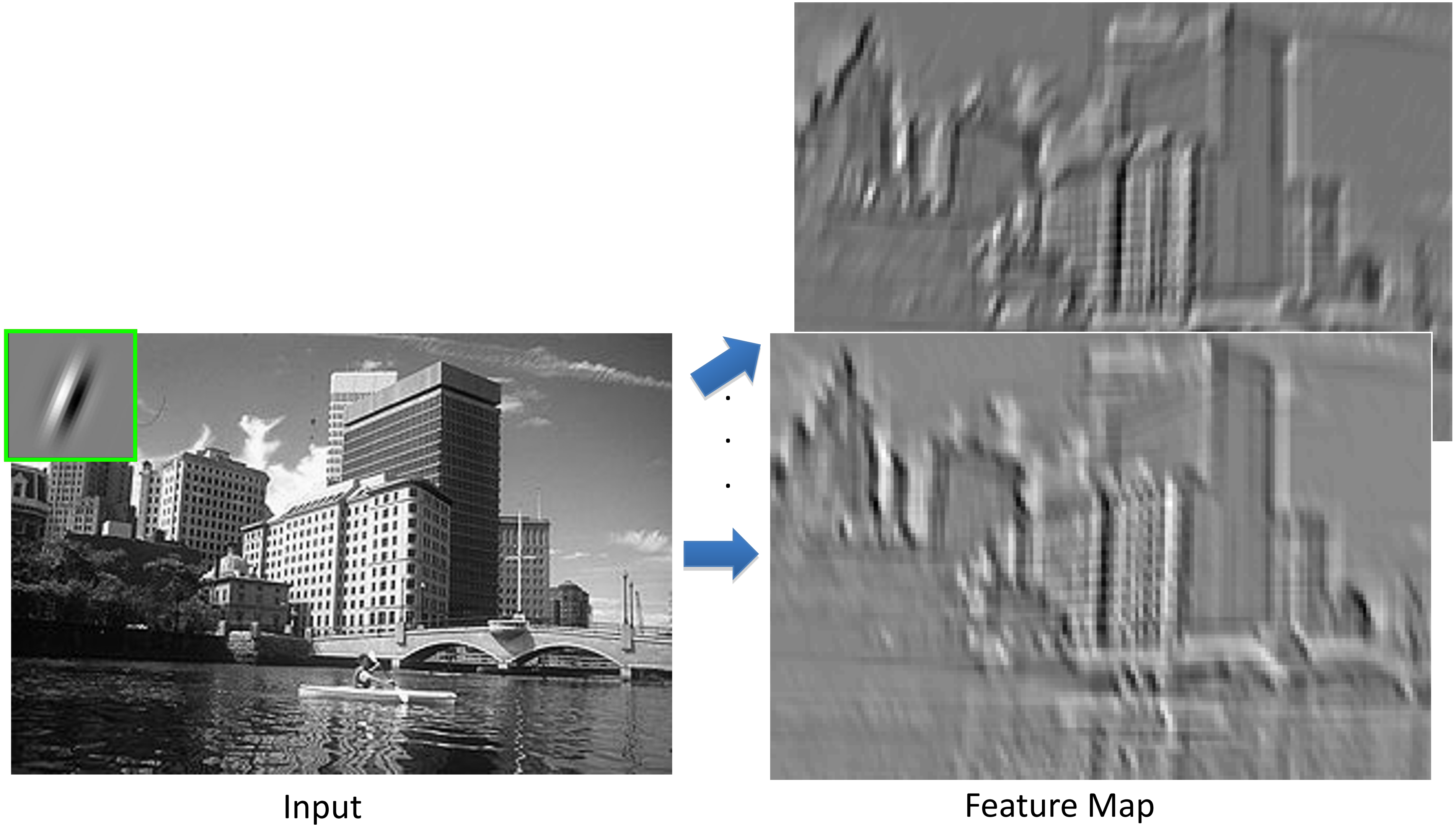
From fully connected to convolutional networks



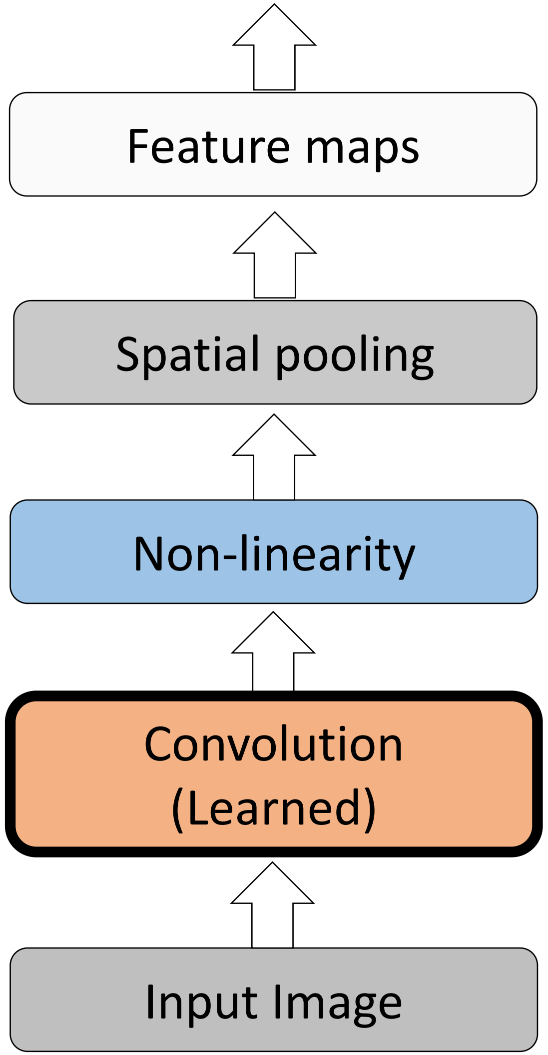
From fully connected to convolutional networks

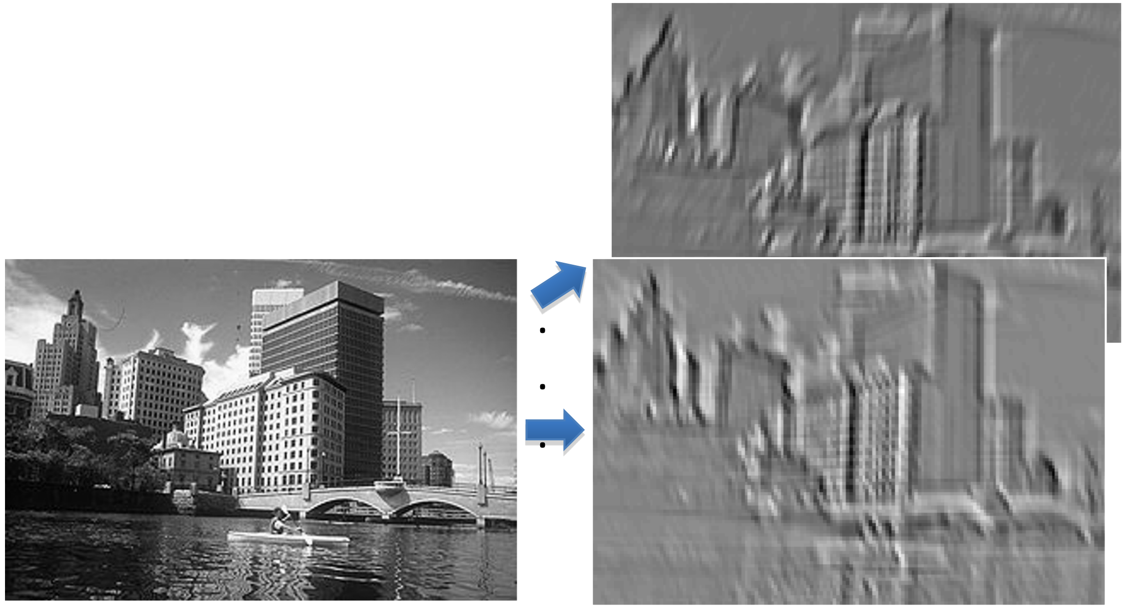


Convolution as feature extraction



Key operations in a CNN





Input

Feature Map

Source: R. Fergus, Y. LeCun

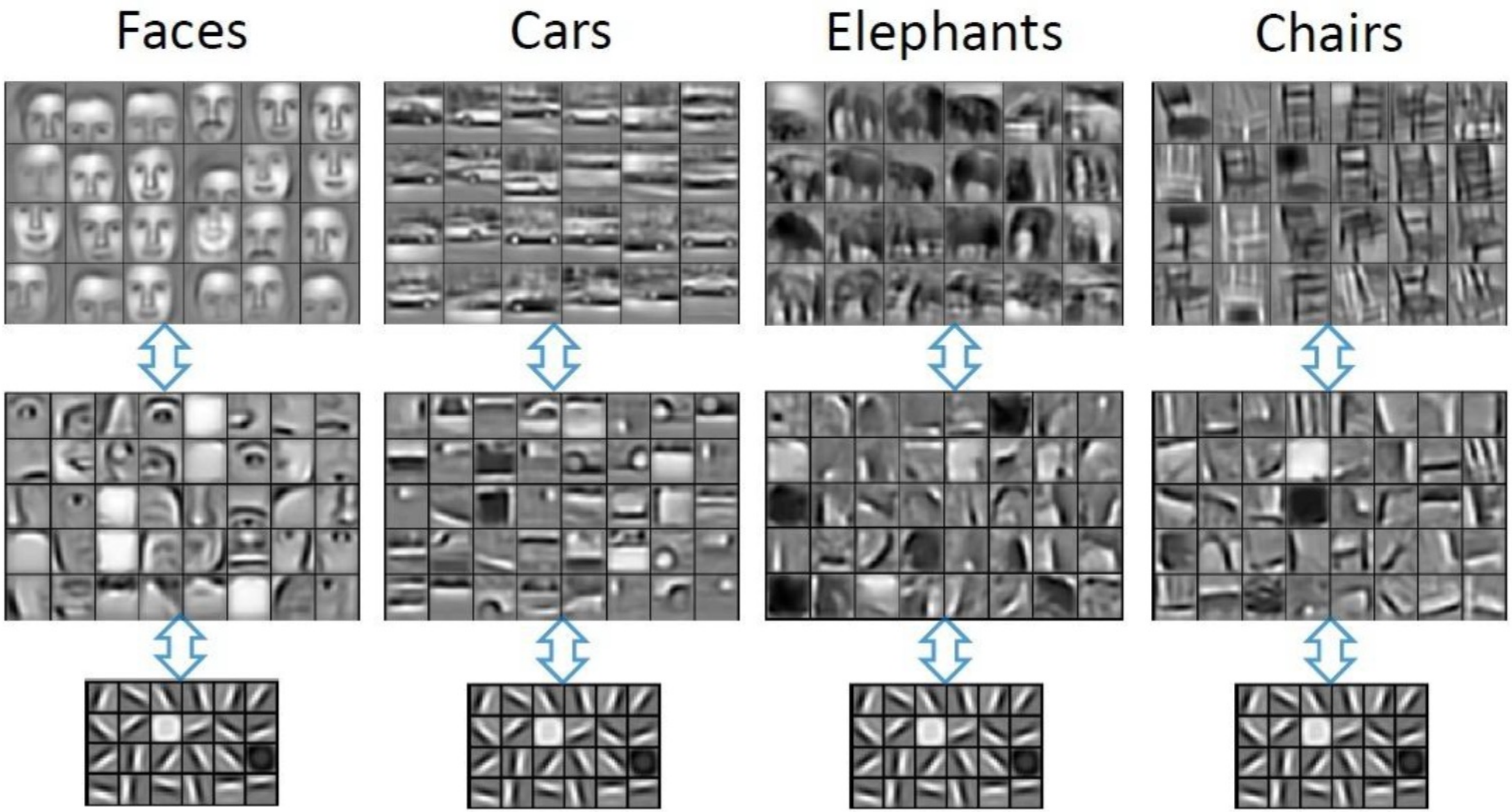
Review: Computer Vision Has "Three Levels"

"There's an edge!"

"There's an object and a background!"

"There's a chair!"

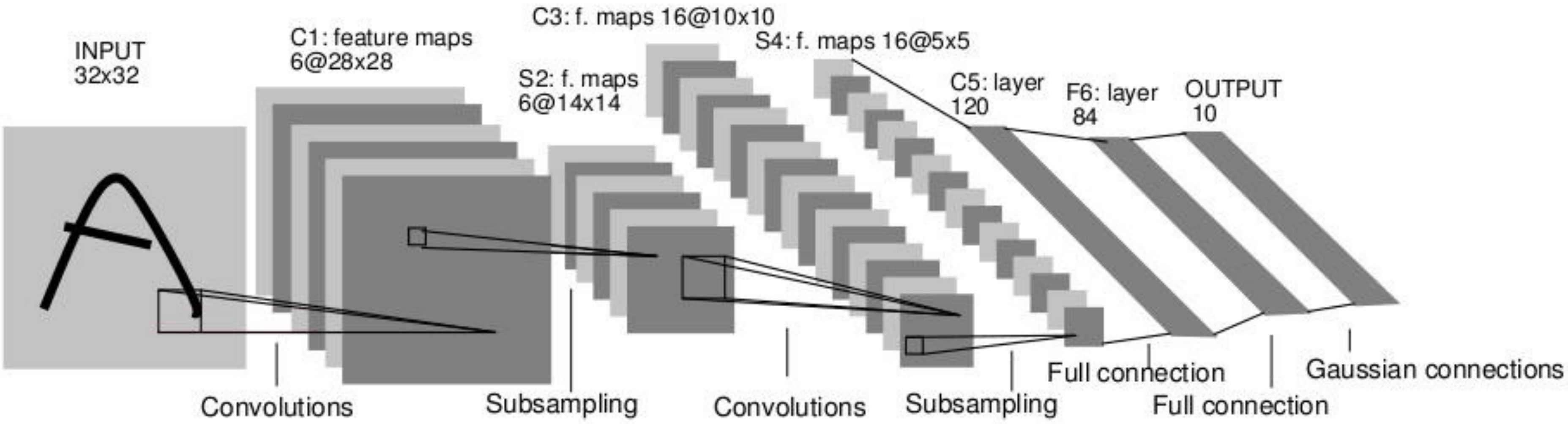
Deep Features (May) Learn Semantic Hierarchy



Popular Backbones: From LeNet to DenseNet

A Remarkable Odyssey to Artificial Intelligence by Human Intelligence

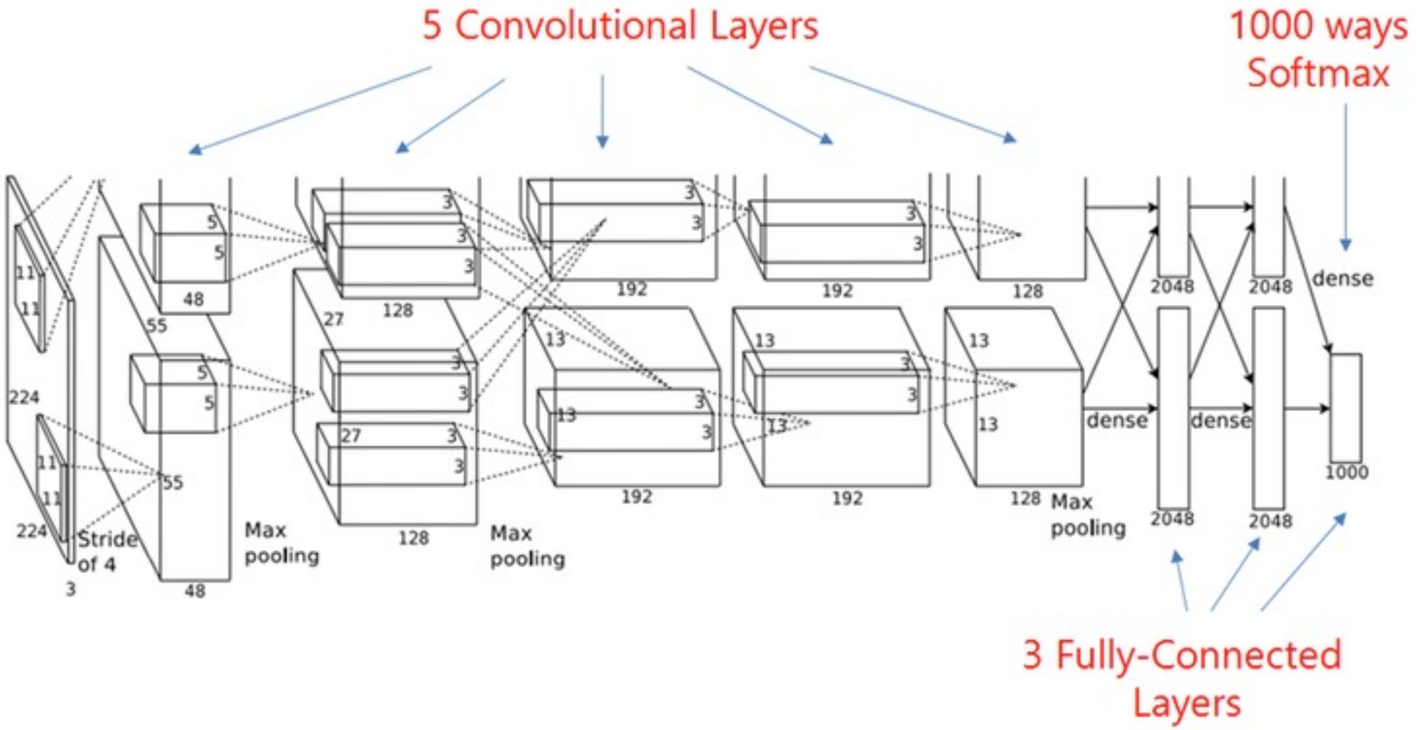
LeNet-5



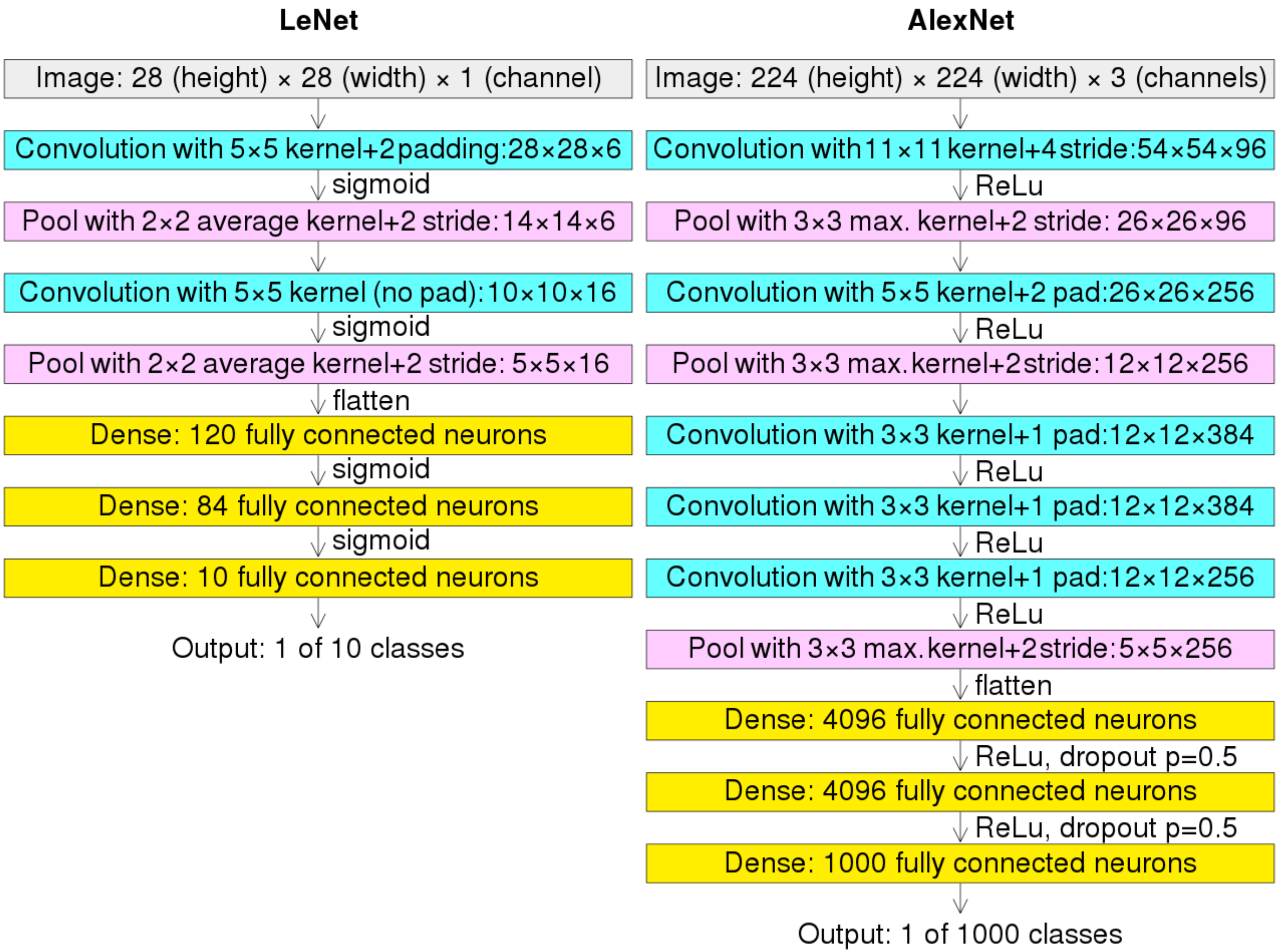
- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples

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

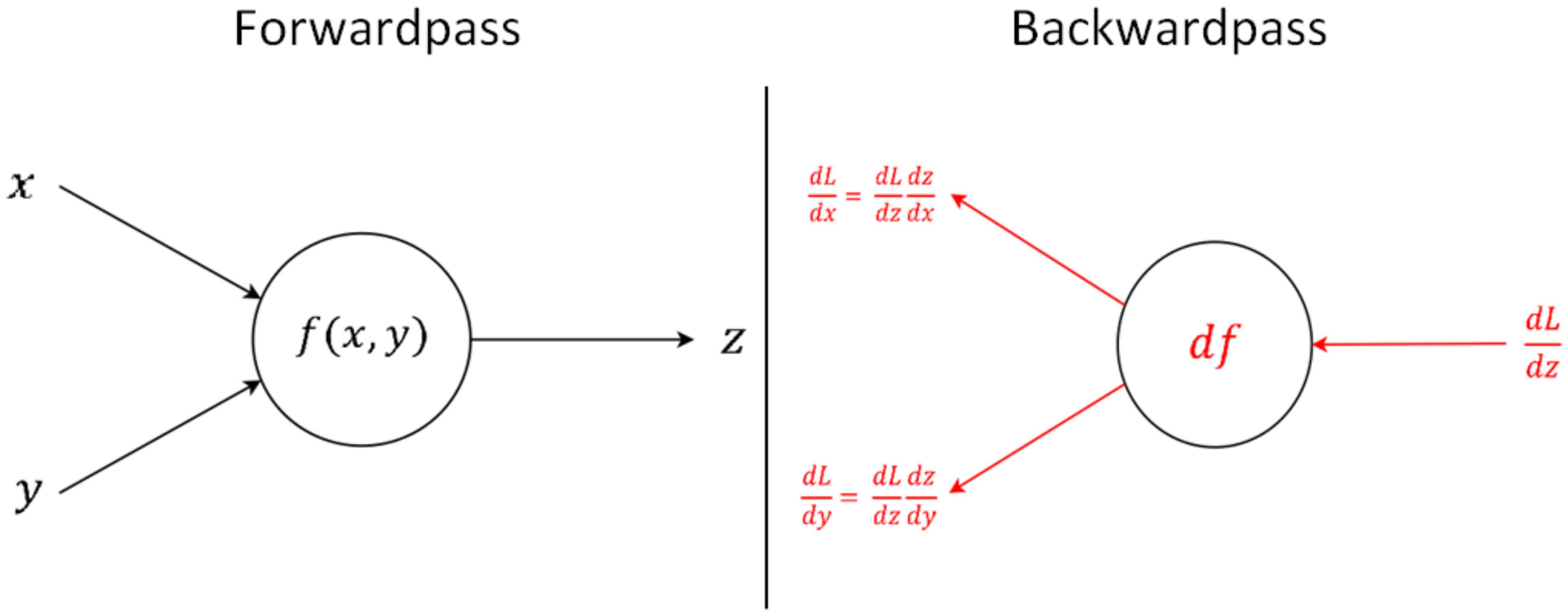
AlexNet, 2012



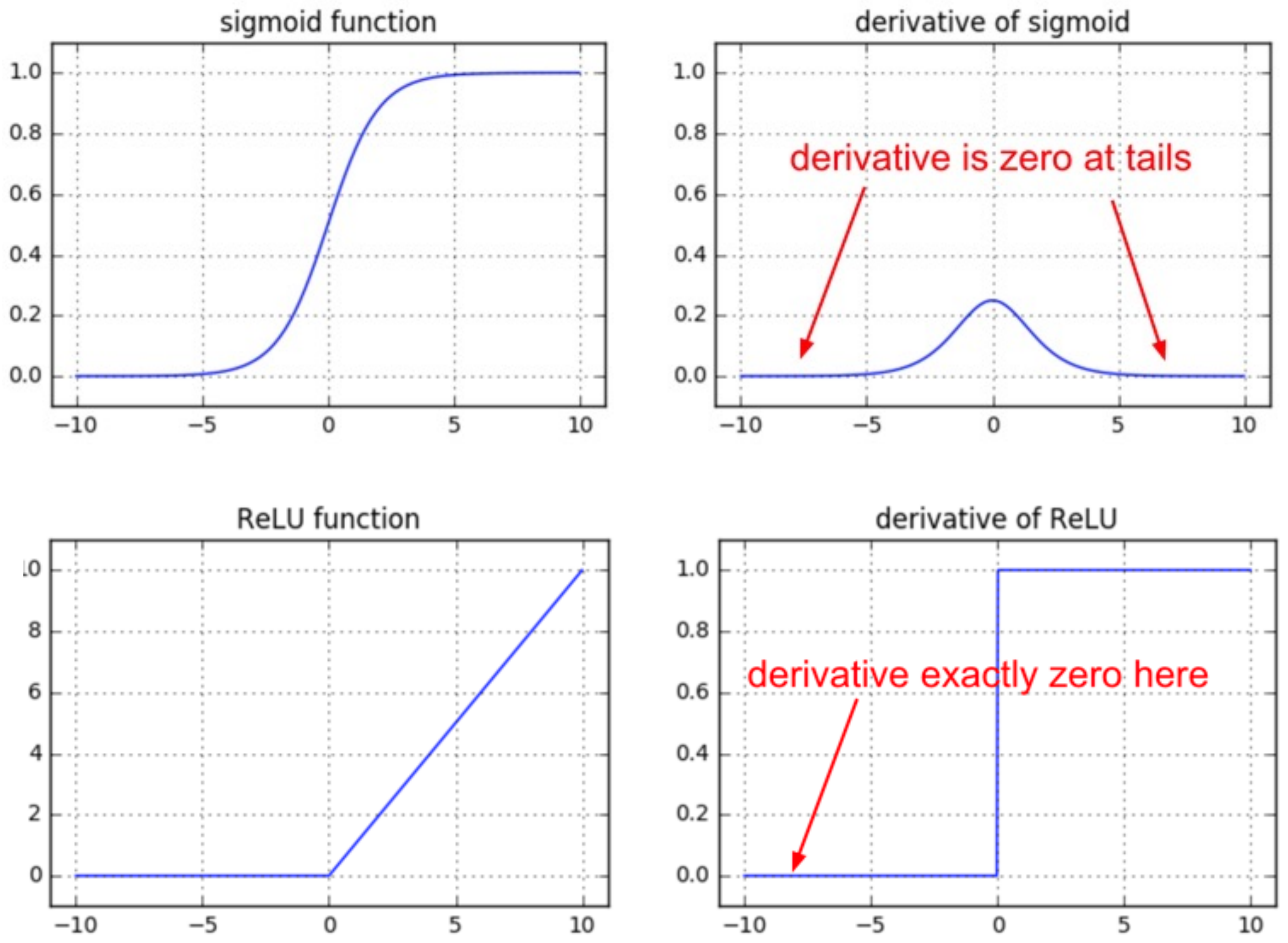
- The **FIRST** winner deep model in computer vision, and one of the most classical choices for domain experts to adapt for their applications
- 5 convolutional layers + 3 fully-connected layers + softmax classifier
- <u>Three Key Design Features:</u> **ReLU, dropout, data augmentation**

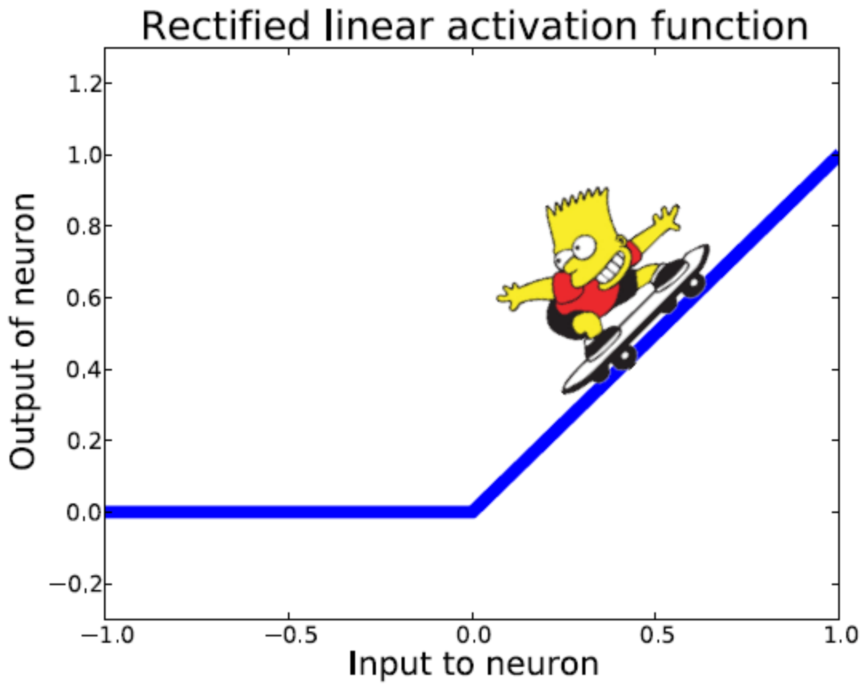

LeNet
AlexNet
Image: 28 (height) × 28 (width) × 1 (channel)
Convolution with 5×5 kernel+2padding:28×28×6
sigmoid
Pool with 2×2 average kernel+2 stride:14×14×6
Convolution with 5×5 kernel (no pad):10×10×16
sigmoid
Pool with 2×2 average kernel+2 stride: 5×5×16
flatten
Dense: 120 fully connected neurons
sigmoid
Dense: 84 fully connected neurons
sigmoid
Dense: 10 fully connected neurons
Output: 1 of 10 classes
Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
Output: 1 of 1000 classes

Recap: “Chain Rule”



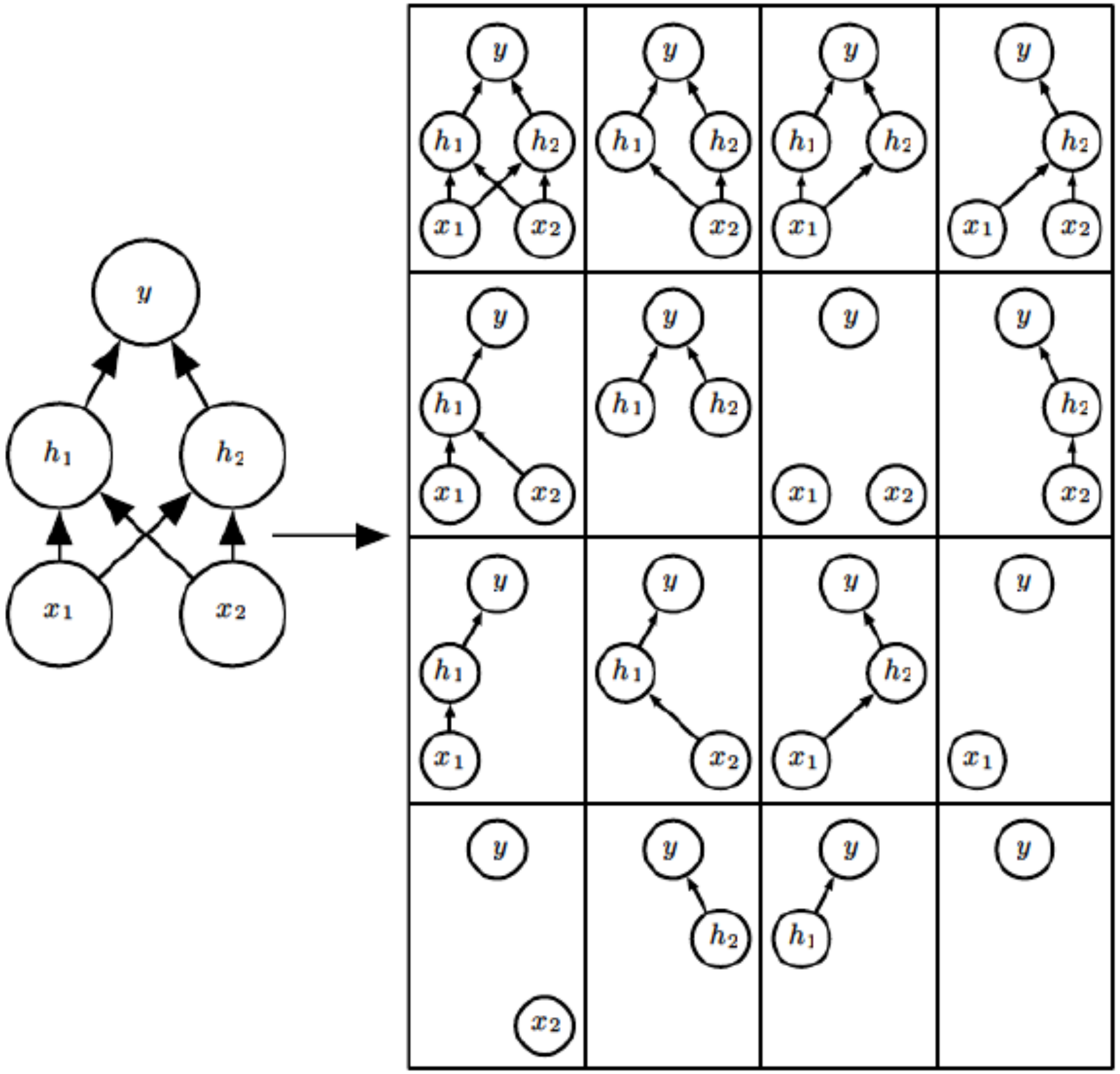
From Sigmoid to ReLU





Dropout

- Randomly select weights to update
  - In each update step, randomly sample a different binary mask to all the input and hidden units
  - Multiple the mask bits with the units and do the update as usual
  - Typical dropout probability: 0.2 for input and 0.5 for hidden units
  - Very useful for FC layers, less for conv layers, not useful in RNNs



Data Augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied -> label preserving
  - **Example:** classifying ‘b’ and ‘d’; ‘6’ and ‘9’

VGG-Net, 2014

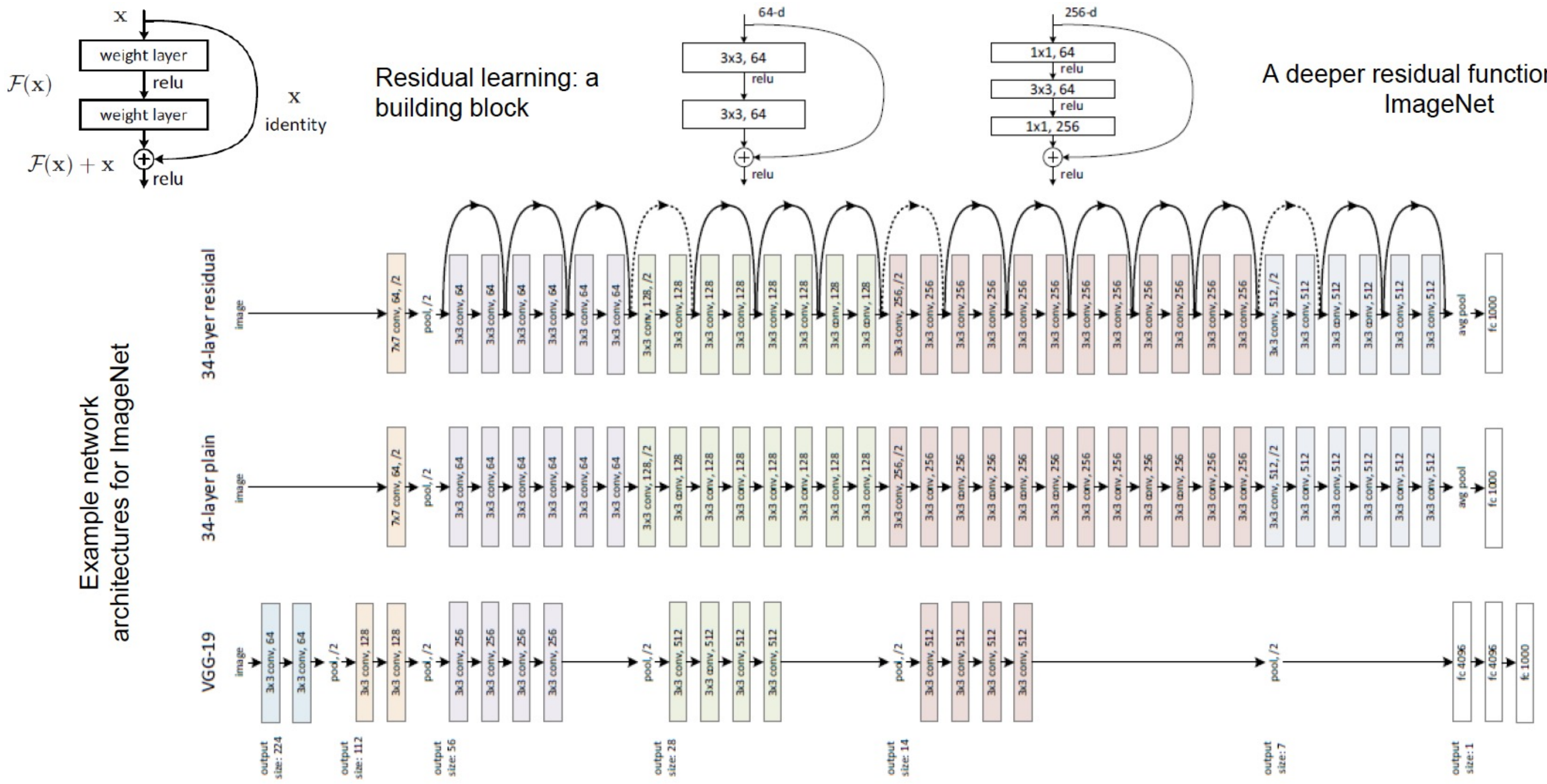
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Key Technical Features:

- Increase depth (up to 19)
- Smaller filter size (3)

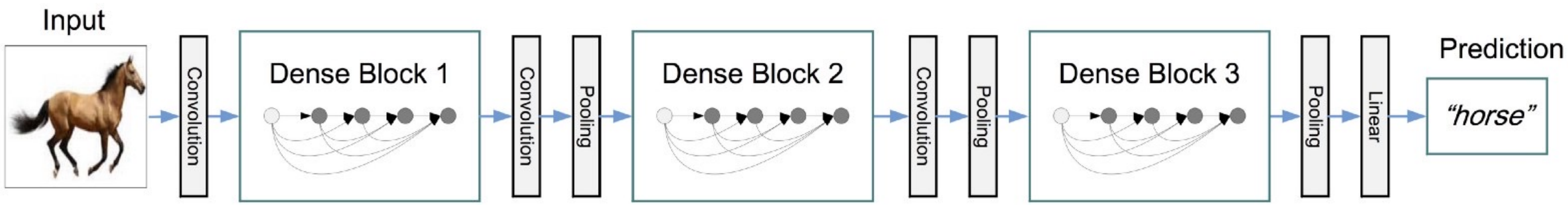
Configurations D and E are widely used for various tasks, called *VGG-16* and *VGG-19*

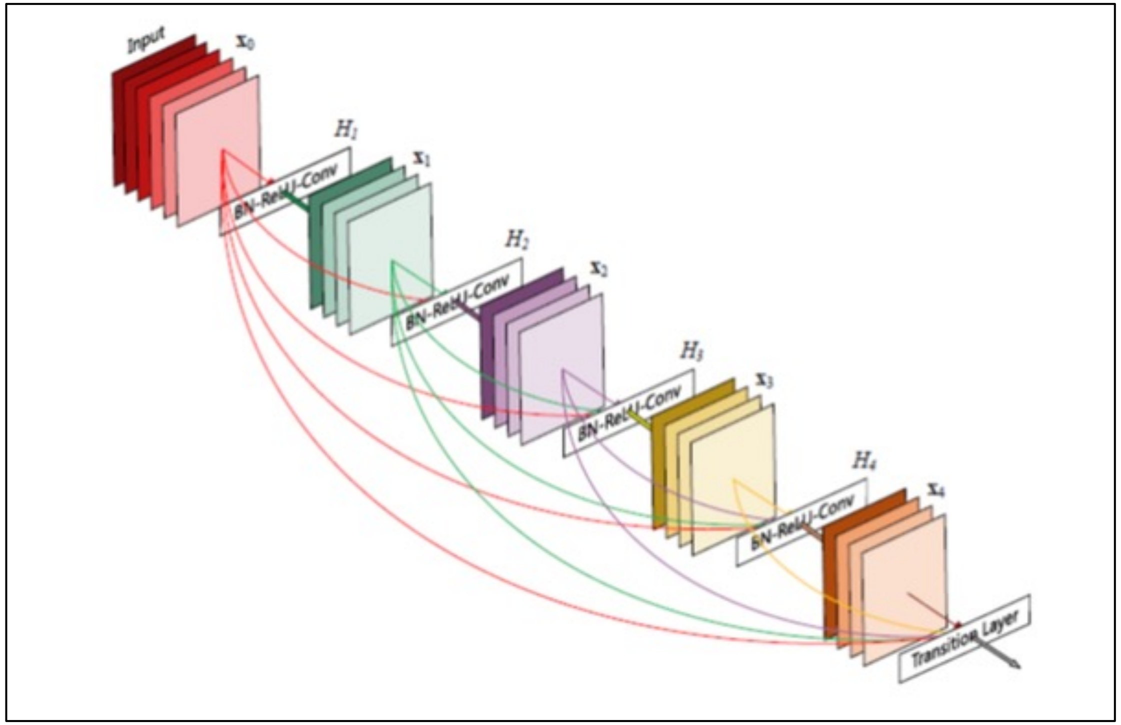
Deep Residual Network (ResNet), 2015



Key Technical Features: skip connections for residual mapping, up to > 1000 layers

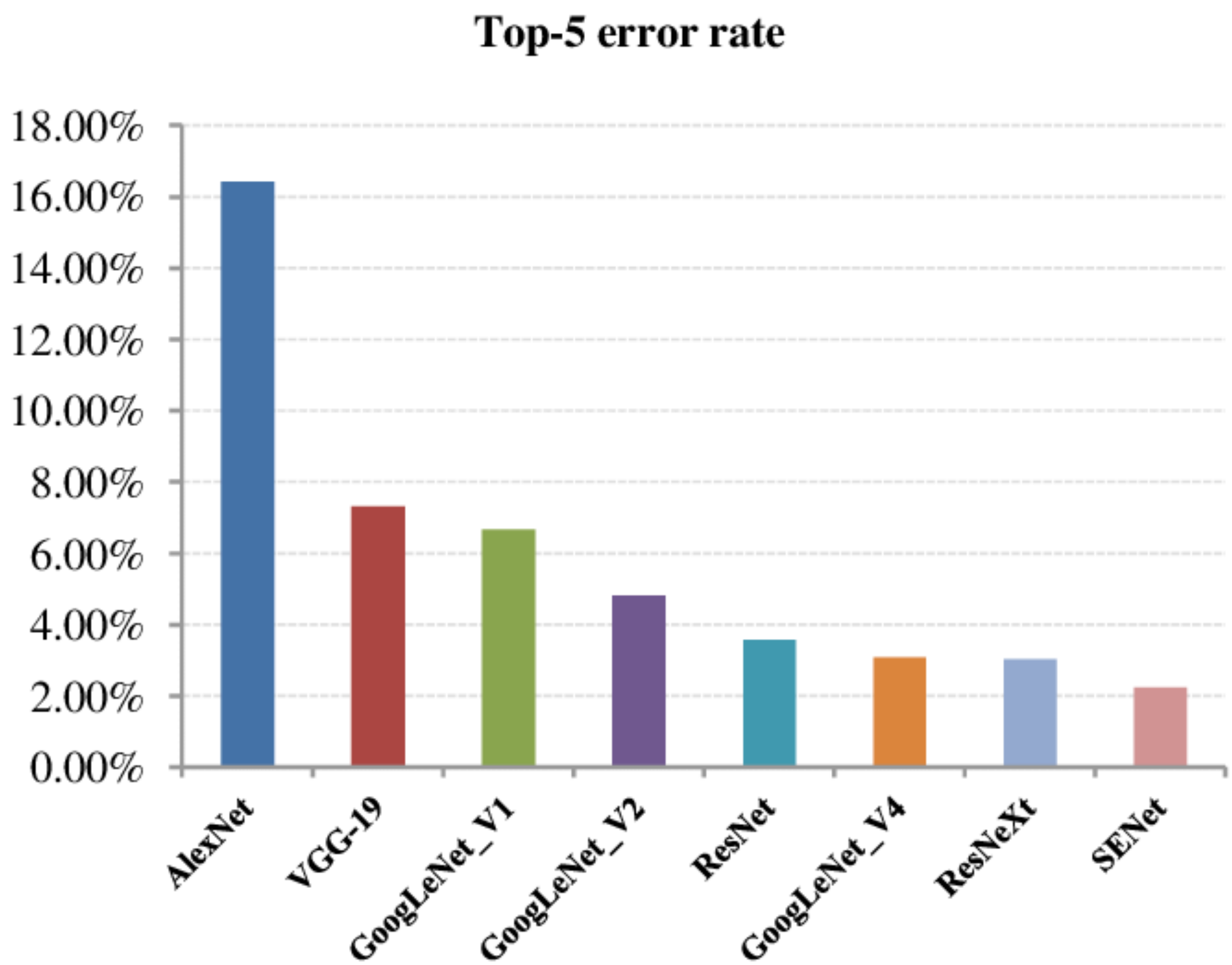
Densely Connected Convolutional Networks (DenseNet), 2017





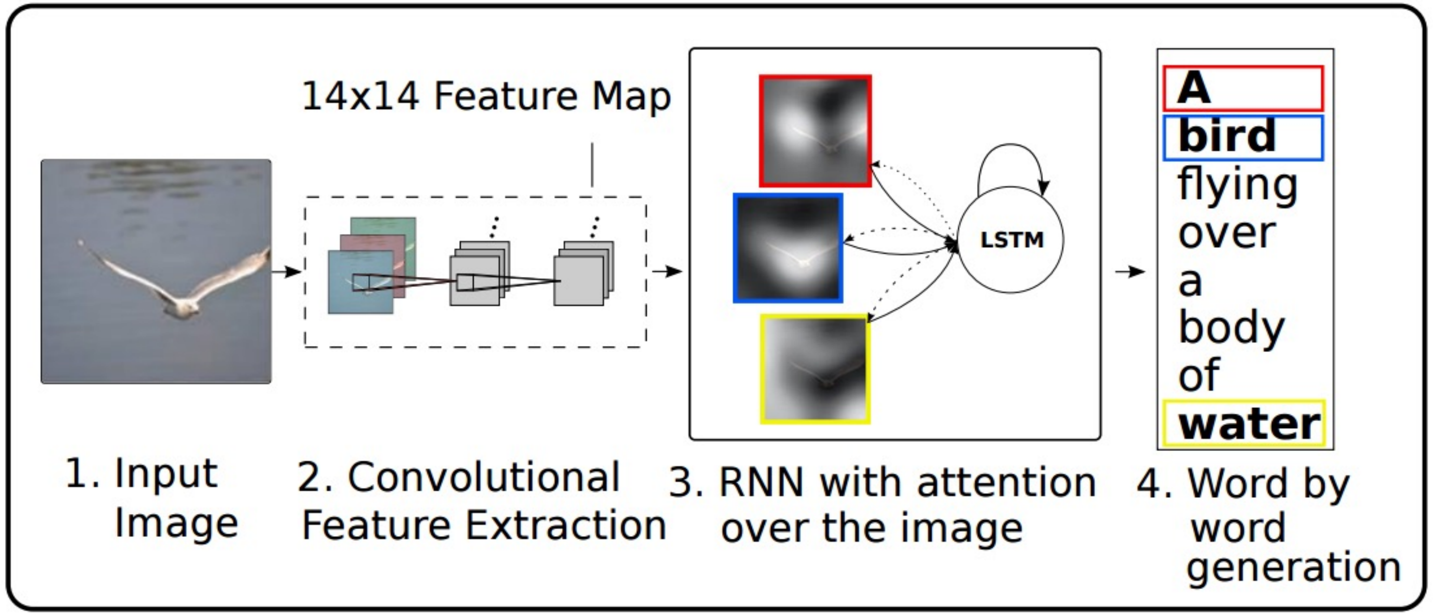
Key Technical Features:

- Finer combination of multi-scale features (or whatever…)



Next Chapter: What is beyond ImageNet classification?

Attention Mechanism



"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", 2015

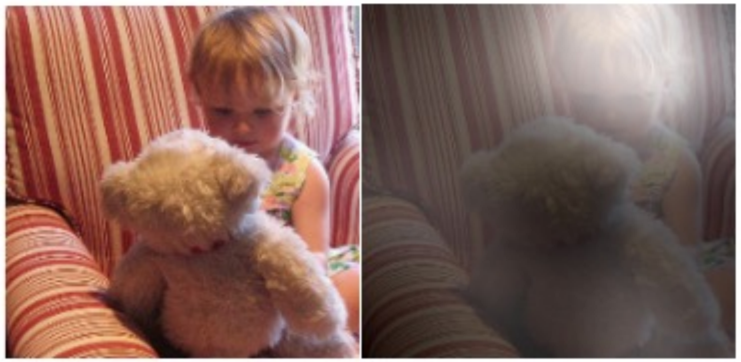
- **Idea is simple:** add a (learned) weighted mask to feature (feature selection)
- Use a feed-forward deep network to extract L feature vectors
- Can use a recurrent network to iteratively update the attention (shown as bright regions) for each output word
- Find meaningful correspondences between words and attentions

Examples of (Input) Visual Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

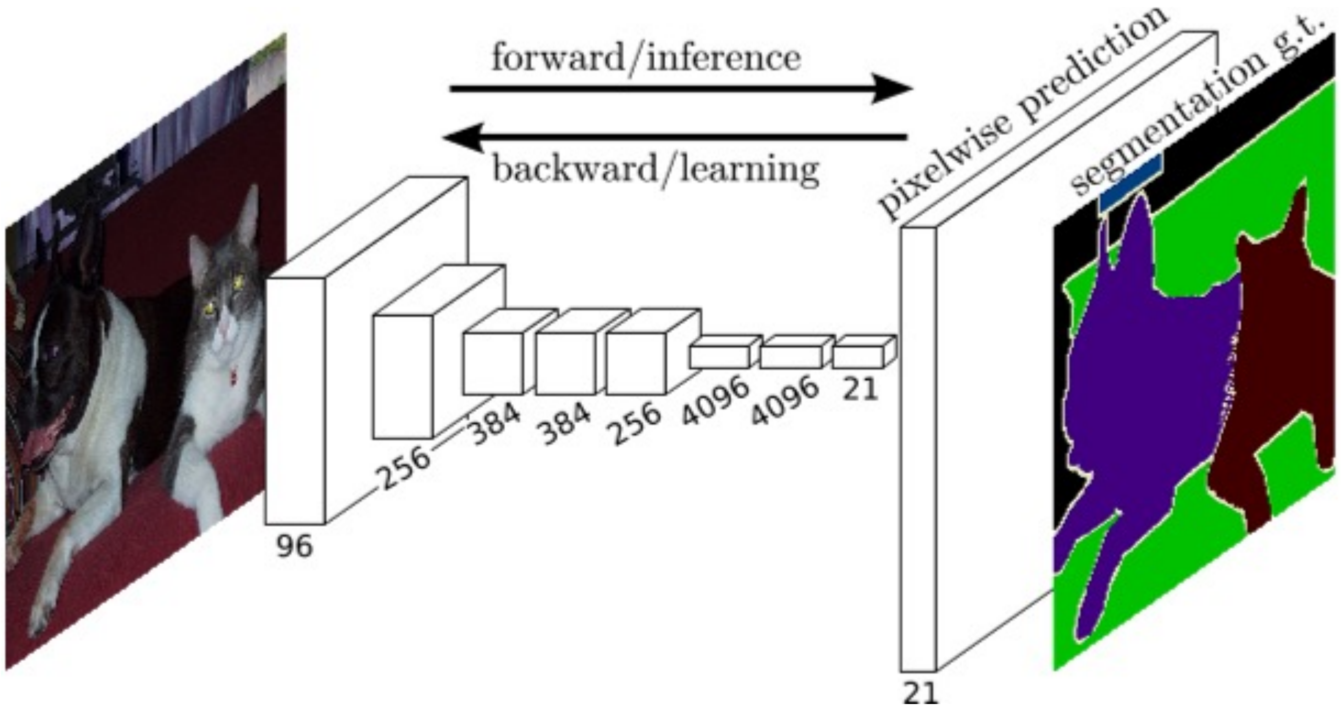


A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

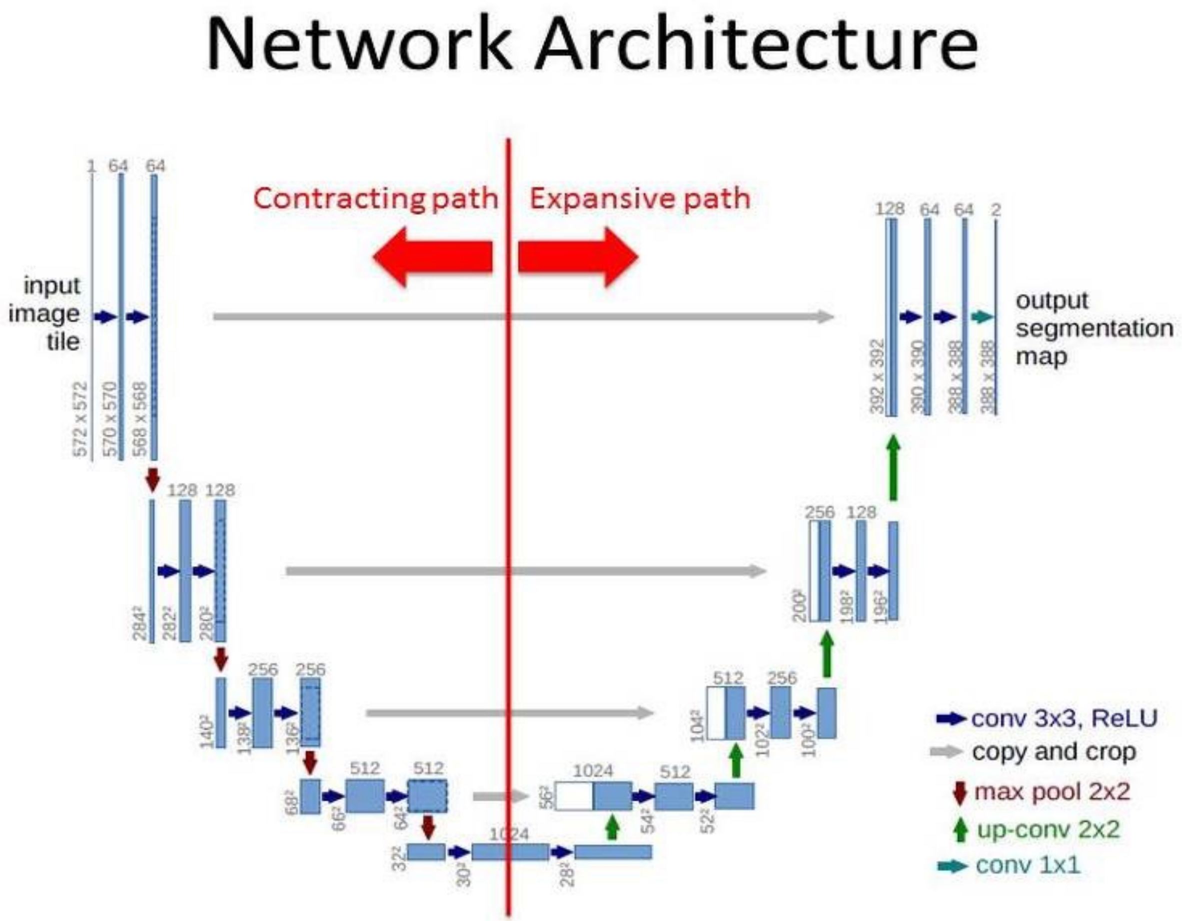
Fully Convolutional Network (FCN), 2014



Key Technical Features:

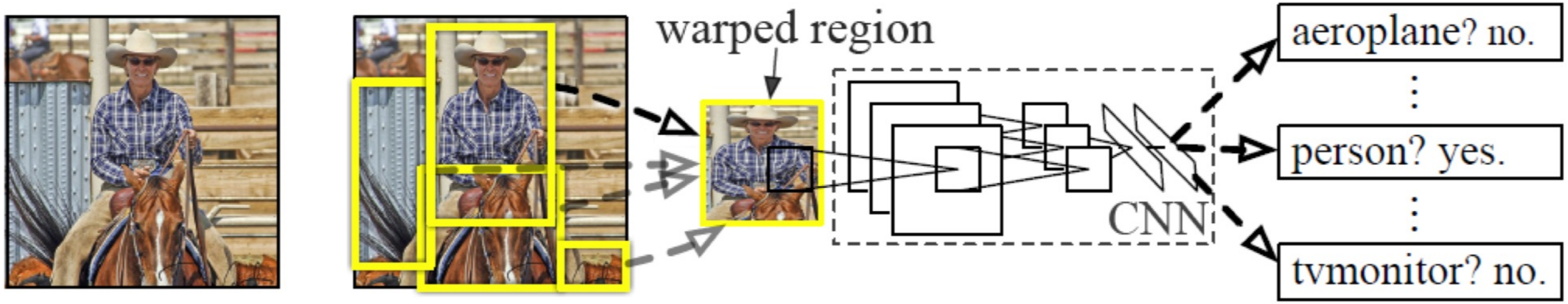
- No fully-connected layer **->** No fixed requirement on input size
- Widely adopted in pixel-to-pixel prediction tasks, e.g., image segmentation

U-Net, 2015



- The architecture consists of a contracting path to capture context
- …and a symmetric expanding path to enable precise localization.
- Also fully convolutional
- Very popular backbone for dense prediction (image segmentation, restoration…)

R-CNN: Region Proposals + CNN



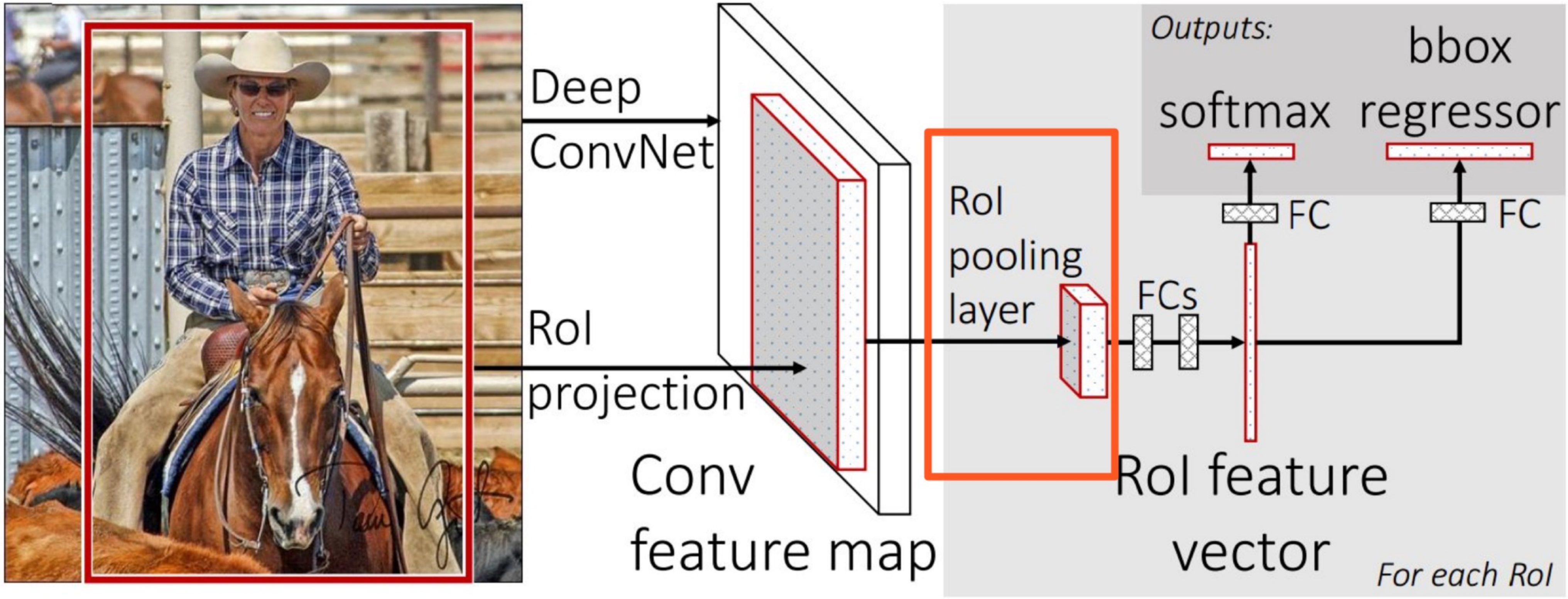
	localization	feature extraction	classification
this paper:	selective search	deep learning CNN	binary linear SVM
alternatives:	objectness, constrained parametric min-cuts, sliding window …	HOG, SIFT, LBP, BoW, DPM …	SVM, Neural networks, Logistic regression …

Fast RCNN

Share convolution layers for proposals from the same image

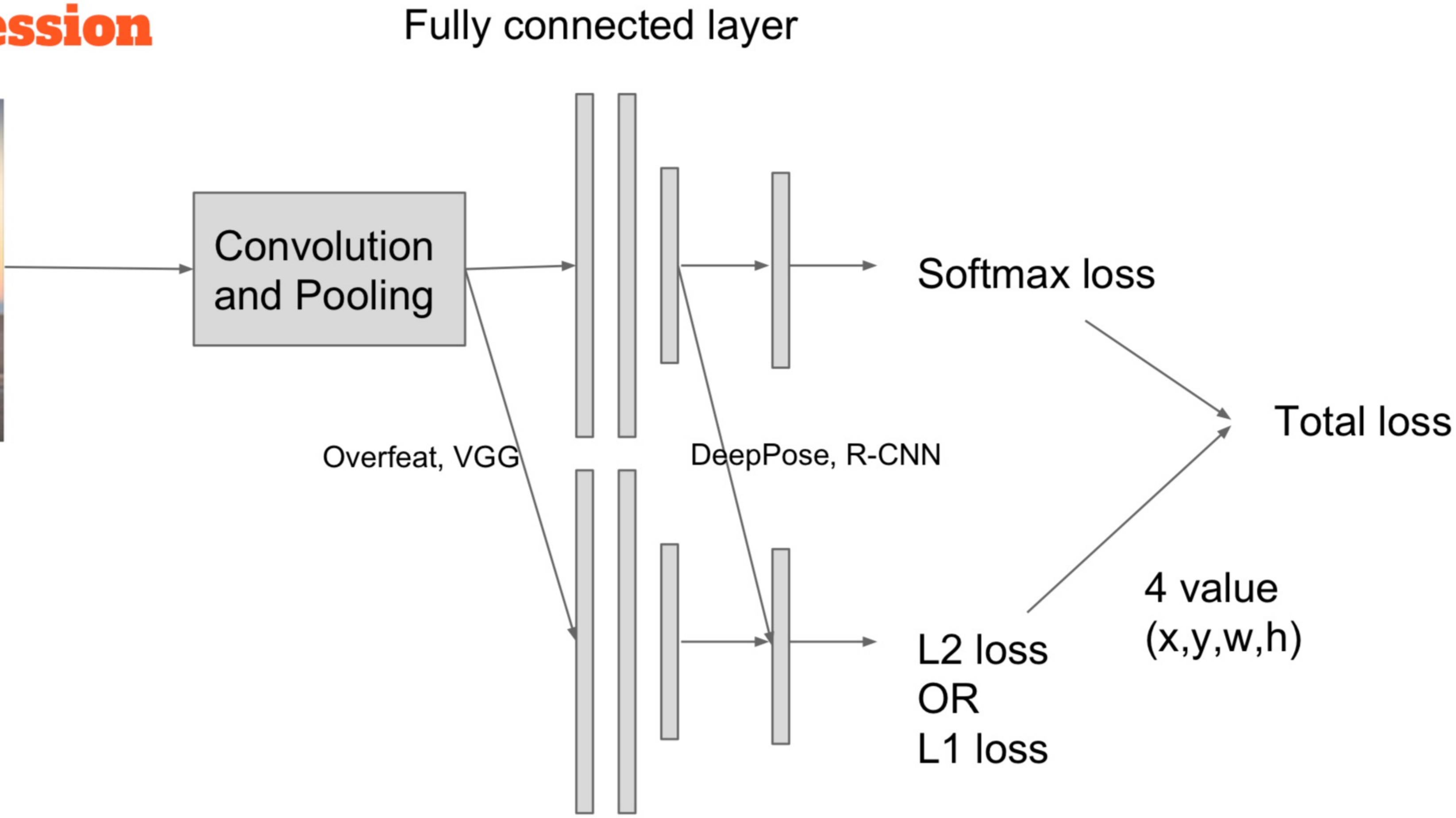
Faster and More accurate than RCNN

ROI Pooling



Fast RCNN

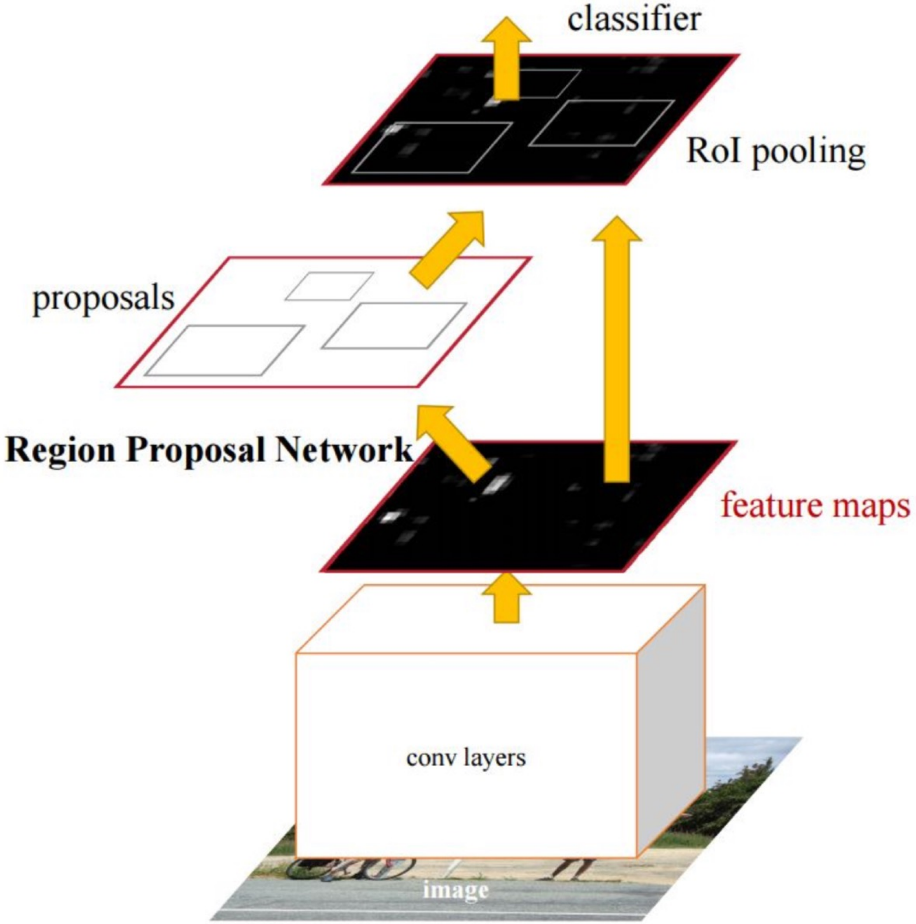
Bounding box regression

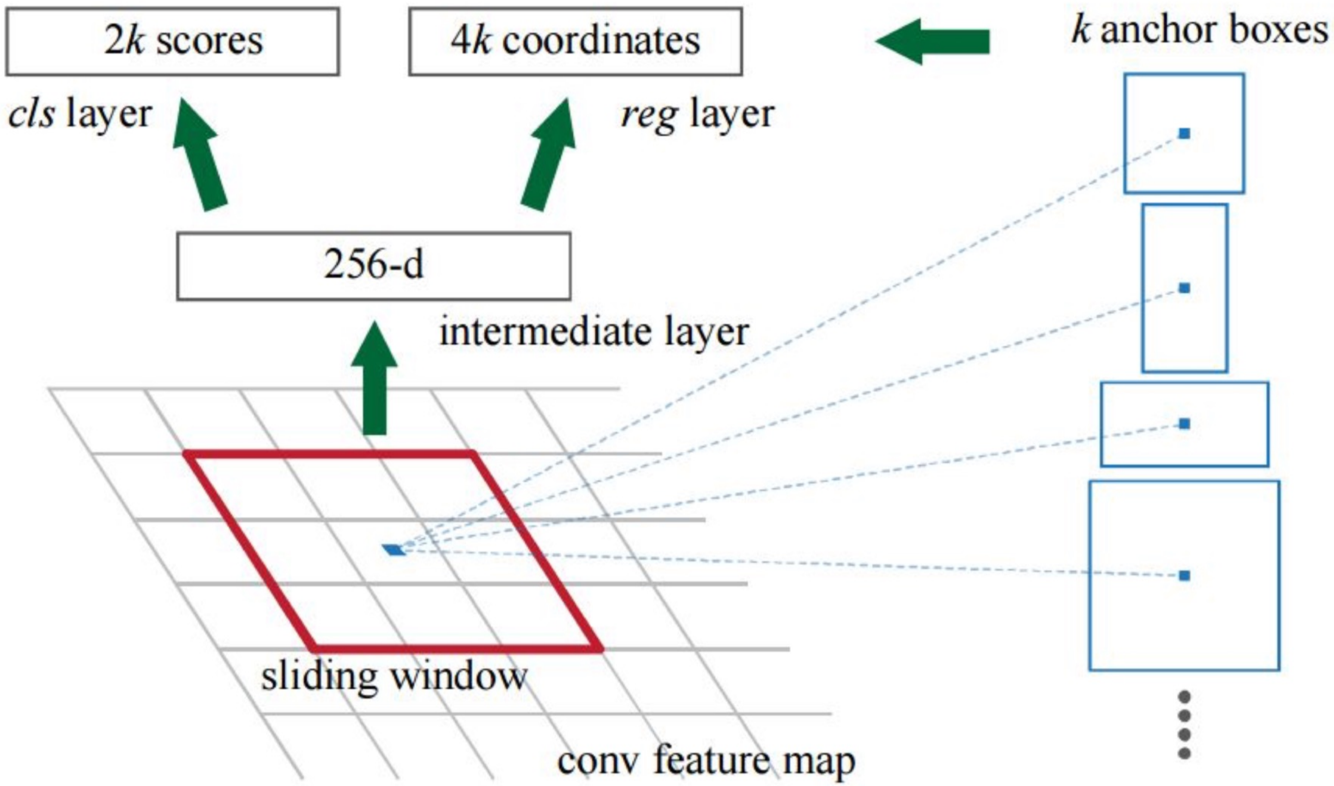


Faster RCNN

Don’t need to have external regional proposals

RPN - Regional Proposal Network





Yolo: You Only Look Once

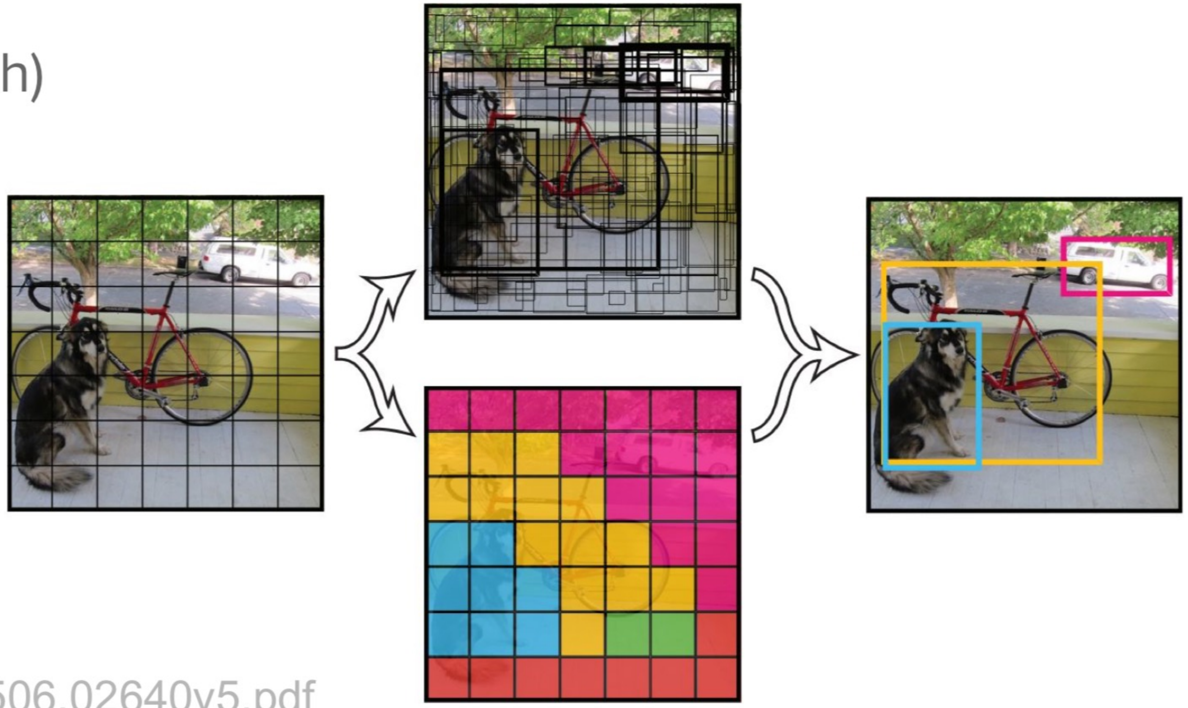
The following predictions are made for each cell in an S x S grid.

C conditional class probabilities **Pr(Class$_i$ | Obj)**

B bounding boxes (4 parameters each)

B confidence scores **Pr(Obj)*IoU**

Output is **S x S x (5B+C)** tensor



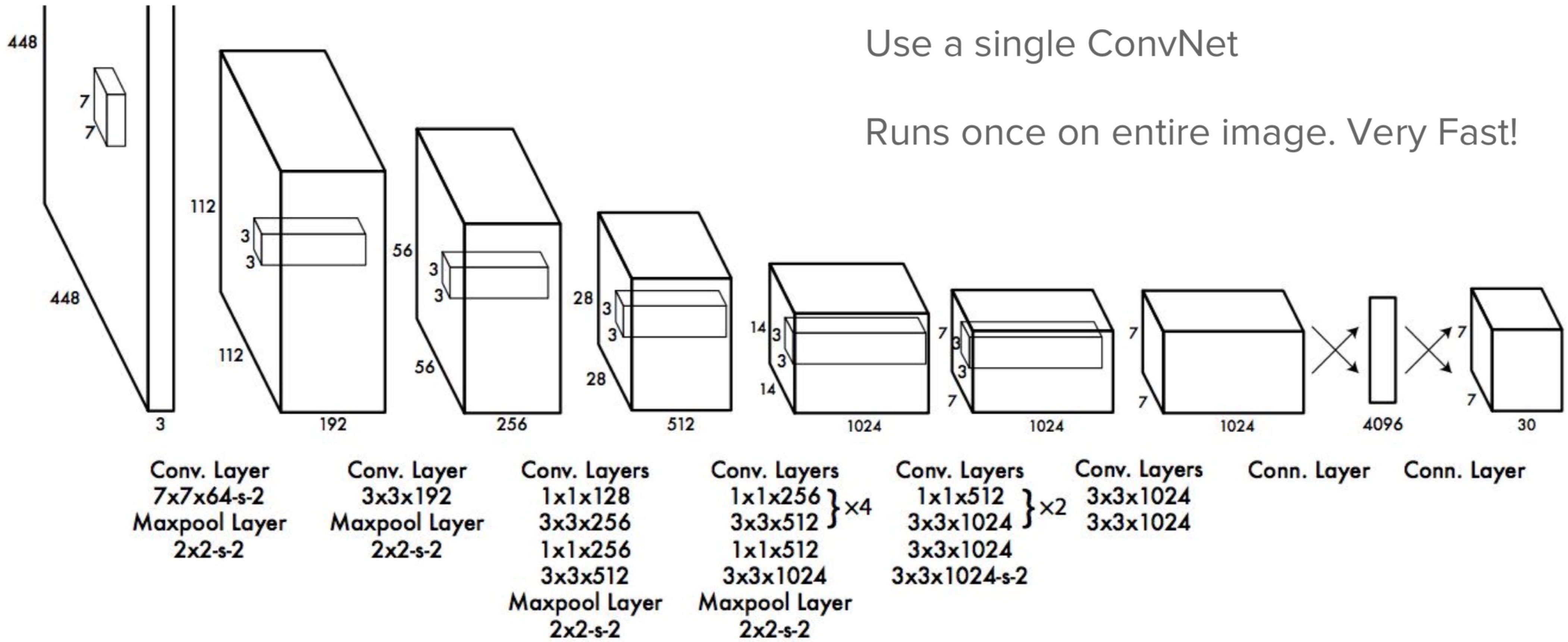
https://arxiv.org/pdf/1506.02640v5.pdf

Yolo: You Only Look Once

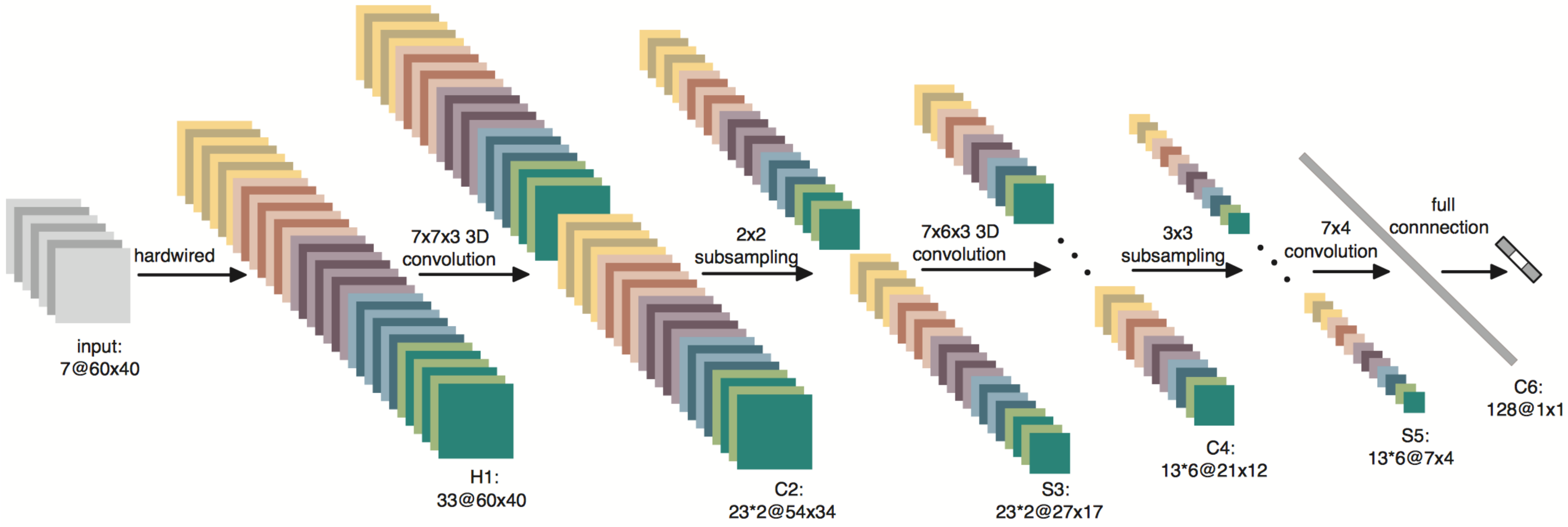
Consider detection a regression problem

Use a single ConvNet

Runs once on entire image. Very Fast!



3D Convolutional Network (3D CNN), 2011

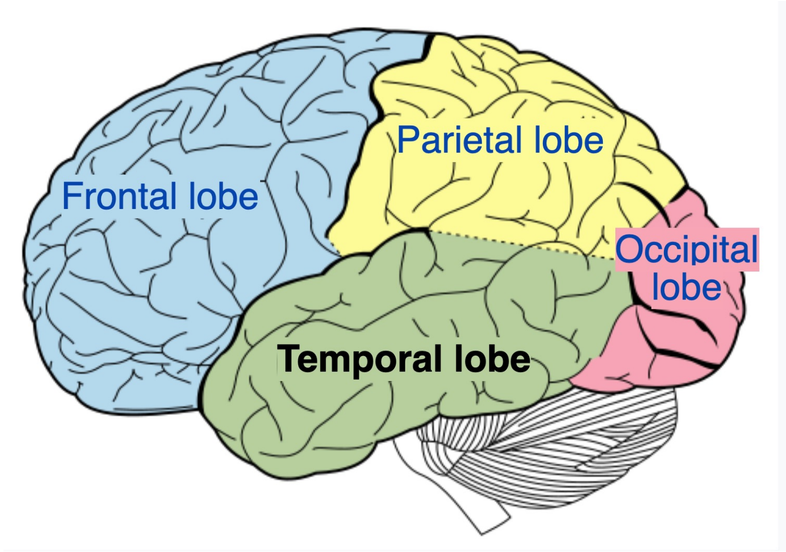


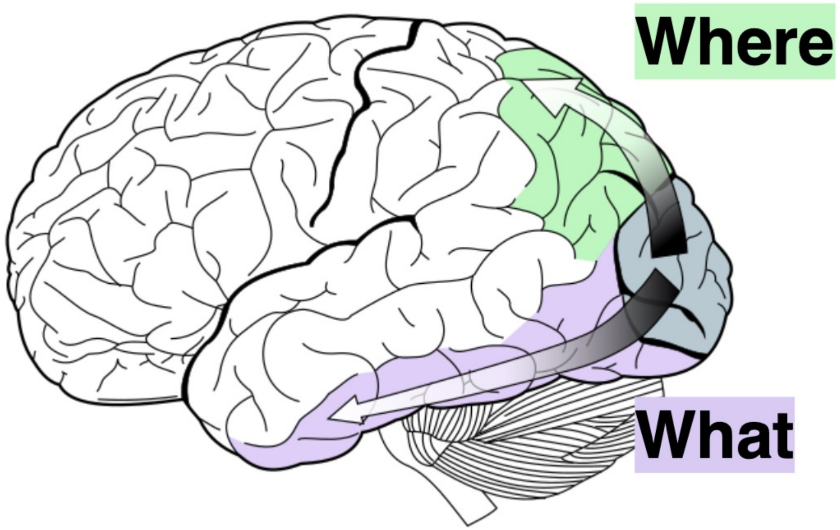
Key Technical Features:

- Going from 2D convolutional filters to 3D filters, to take temporal coherence into consideration

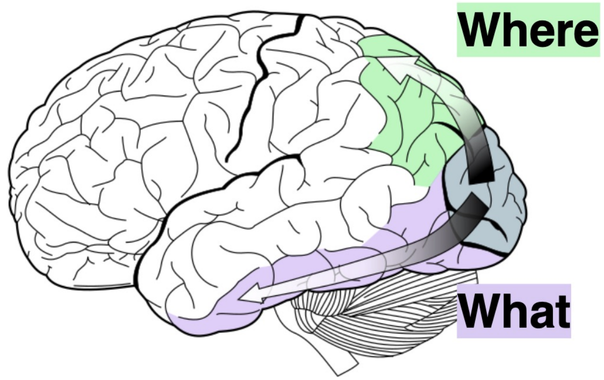
More Efficient Design?

- “**Two-streams hypothesis**” for human vision
  - The **dorsal stream** involves in the guidance of actions and recognizing where objects are in space. It contains a detailed map of the visual field. and detects & analyzes location movements
  - The **ventral stream** is associated with object recognition and form representation. Also described as the “what” stream, it has strong connections to the dorsal stream and other brain regions controlling memory or emotion
- **Long story short:** human brains use two relatively independent systems to recognize objects and to record temporal movements.





Two Stream Network, 2014



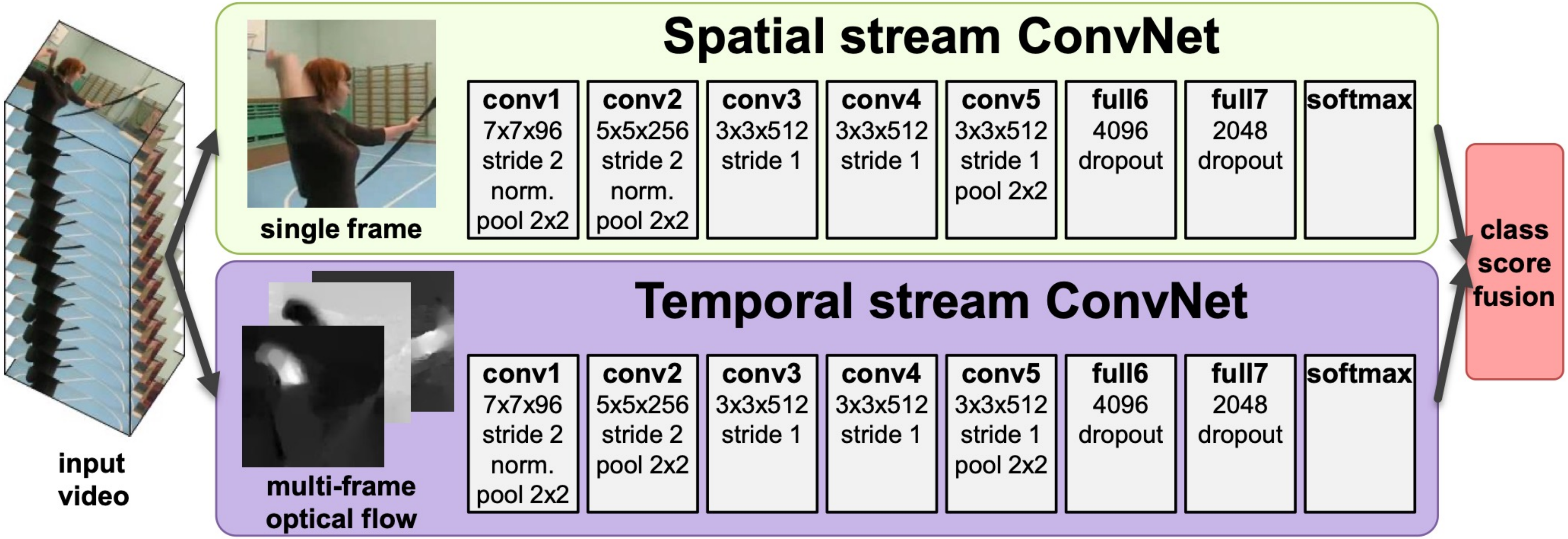
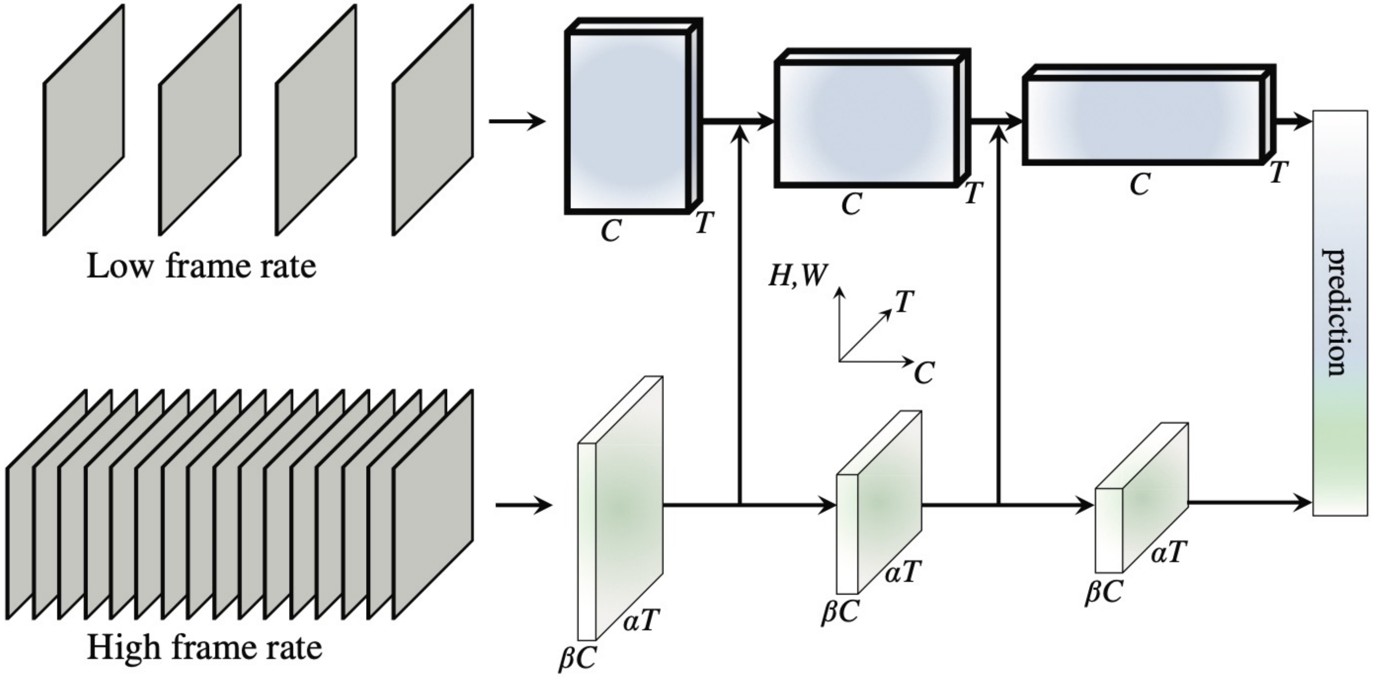


Figure 1: **Two-stream architecture for video classification.**

Slow-Fast Network, 2019

A state-of-the-art two-stream model with

- *(i) a Slow pathway, operating at low frame rate, to capture spatial semantics*
- *(ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution.*



Optimization Algorithms

Where the magic happens

Gradient Descent (GD)

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ
1: **while** stopping criteria not met **do**
2: Compute gradient estimate over N examples:
3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
5: **end while**

- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ

while stopping criteria not met **do**
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_{\theta} L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$
end while

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

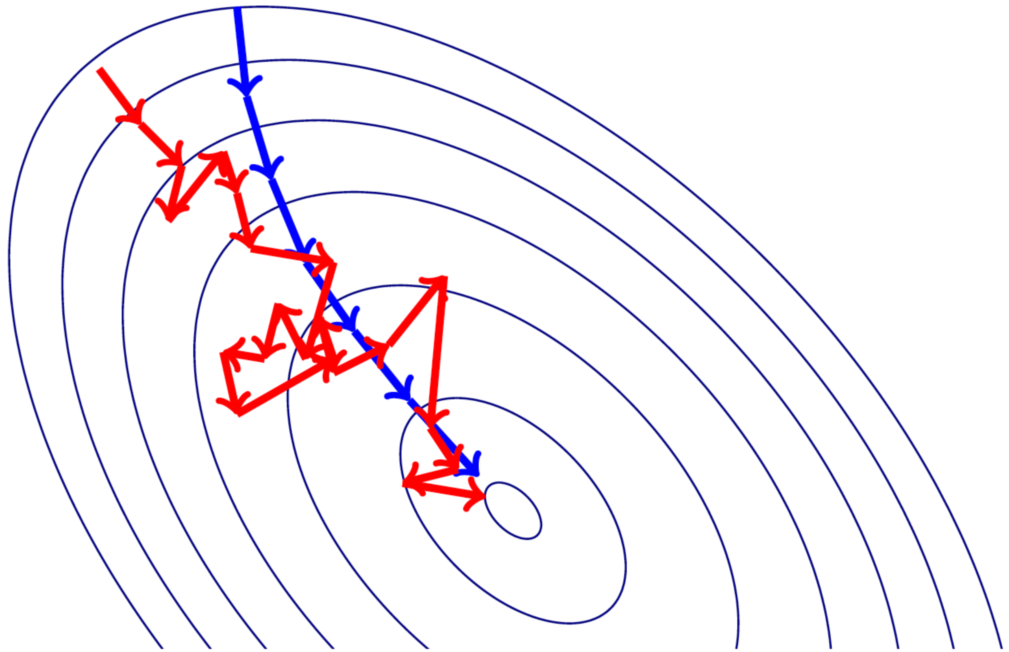
GD versus SGD

- Batch Gradient Descent:

$$\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$$
$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$

- SGD:

$$\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$$
$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$



Minibatch

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches (In theory, growingly larger)

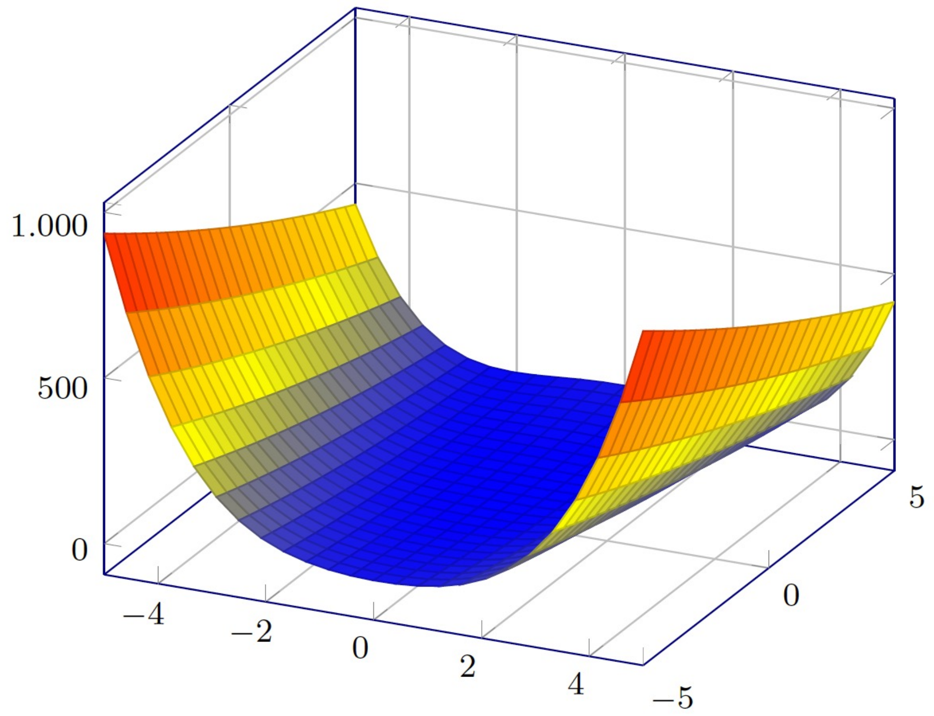
- Advantage: Computation time per update does not depend on number of training examples.
- This allows convergence on extremely large datasets

- **The larger MB size the better (only if you can)!!**

"Large Scale Learning with Stochastic Gradient Descent", Leon Bottou.

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - Small but consistent gradients
  - Noisy gradients



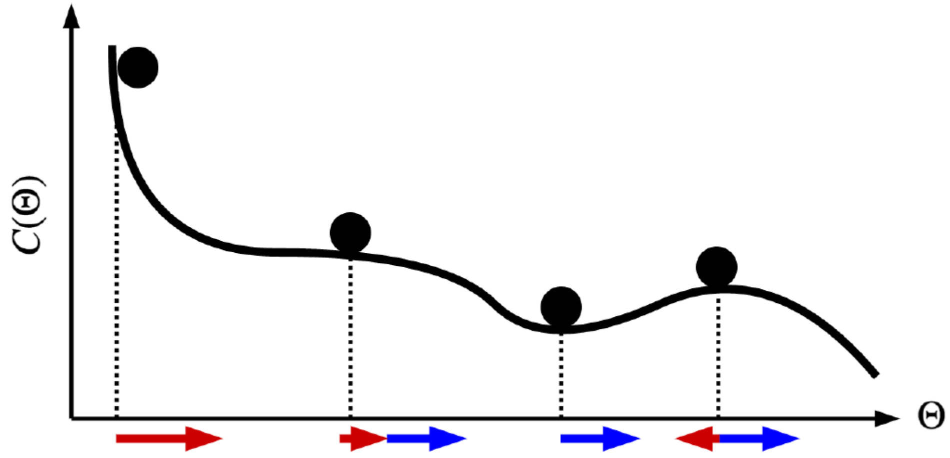
- Gradient Descent would move quickly down the walls, but very slowly through the valley floor

Momentum

- Update rule in SGD:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta \boldsymbol{g}^{(t)}$$

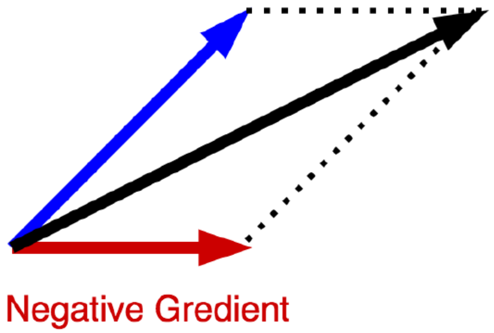
where $\boldsymbol{g}^{(t)} = \nabla_{\Theta} C(\Theta^{(t)})$
  - Gets stuck in local minima or saddle points



- Momentum: make the same movement $\boldsymbol{v}^{(t)}$ in the last iteration, corrected by negative gradient:

$$\boldsymbol{v}^{(t+1)} \leftarrow \lambda \boldsymbol{v}^{(t)} - (1-\lambda) \boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \eta \boldsymbol{v}^{(t+1)}$$



- $\boldsymbol{v}^{(t)}$ is a moving average of $-\boldsymbol{g}^{(t)}$

Adaptive Learning Rate Optimization

- Popular Solver Examples: AdGrad, RMSProp, Adam

$$\text{SGD: } \theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$$

$$\text{Momentum: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

$$\text{Nesterov: } \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right) \text{ then } \theta \leftarrow \theta + \mathbf{v}$$

$$\text{AdaGrad: } \mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \text{ then } \Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{RMSProp: } \mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}} \text{ then } \Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

$$\text{Adam: } \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t} \text{ then } \Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

Batch Normalization

- In ML, we assume future data will be drawn from same probability distribution as training data
- For a hidden layer, after training, the earlier layers have new weights and hence may generate a new distribution for the next hidden layer
- We want to reduce this internal covariate shift for the benefit of later layers

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

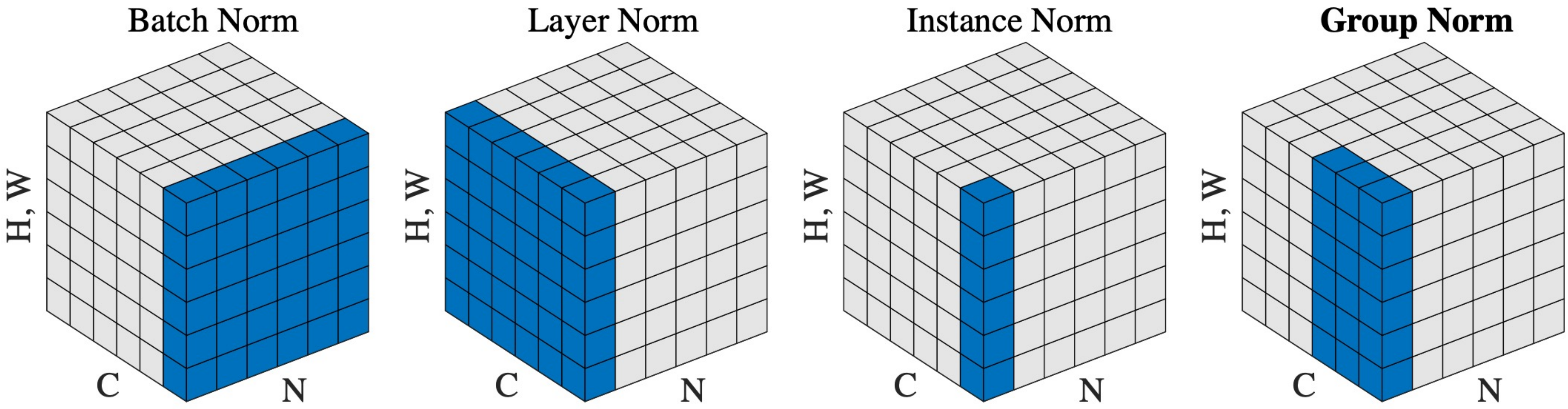
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Many Normalization Schemes...



Comparing Popular Normalization methods. Each subplot shows a feature map tensor, with *N* as the batch axis, *C* as the channel axis, and (*H, W*) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Weight Initialization

- All Zero Initialization: Terribly Wrong!
  - If every neuron in the network computes the same output, then they will also all compute the same gradients during back-propagation and undergo the exact same parameter updates.
  - Need “break the symmetry”

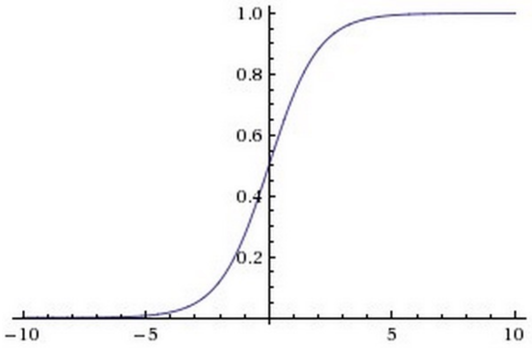
- Small Random Initialization is the standard practice
- **Current recommendation for initializing CNNs with RELU: Why?**

```
w = np.random.randn(n) * sqrt(2.0/n)
```

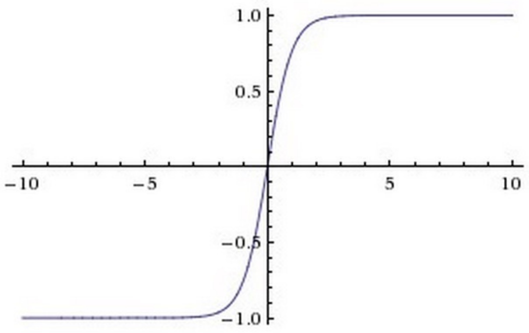
- “randn”: Gaussian; “n”: the number of inputs for current layer.

- For general NNs, layer-wise pre-training is safe.
- Even safer: start from a pre-trained model

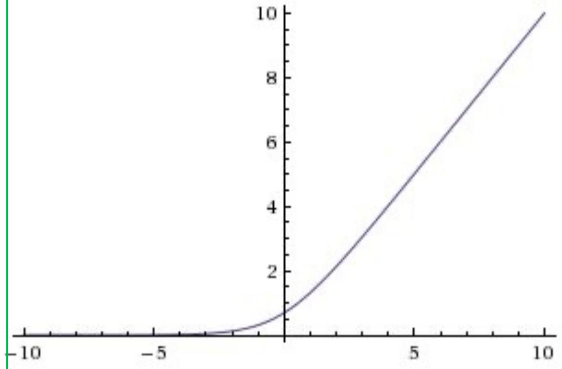
Choice of Activation Functions



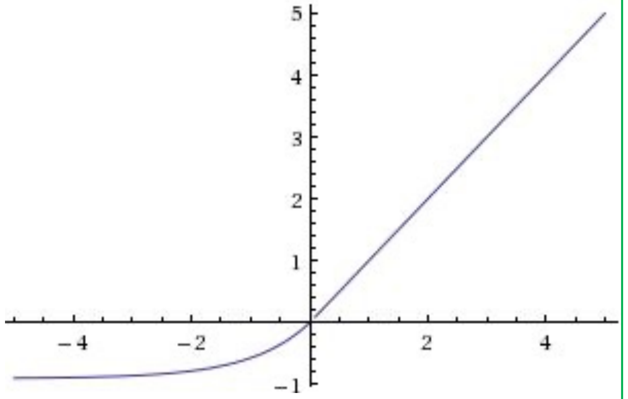
Sigmoid



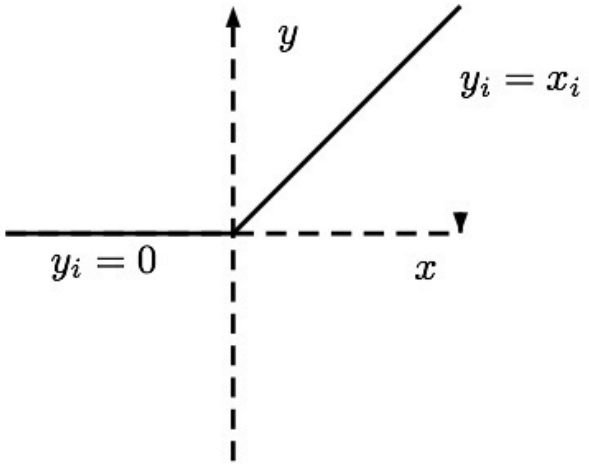
Tanh



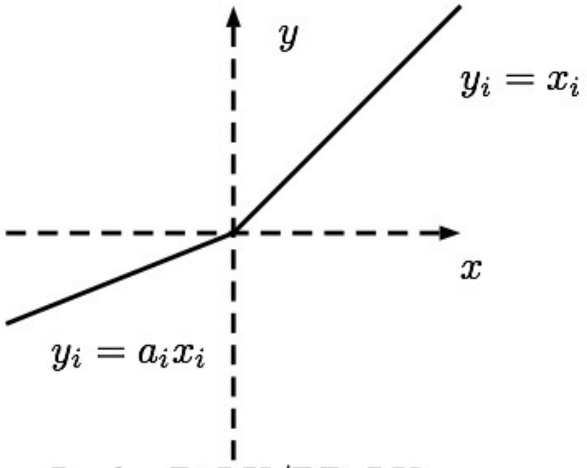
Softplus



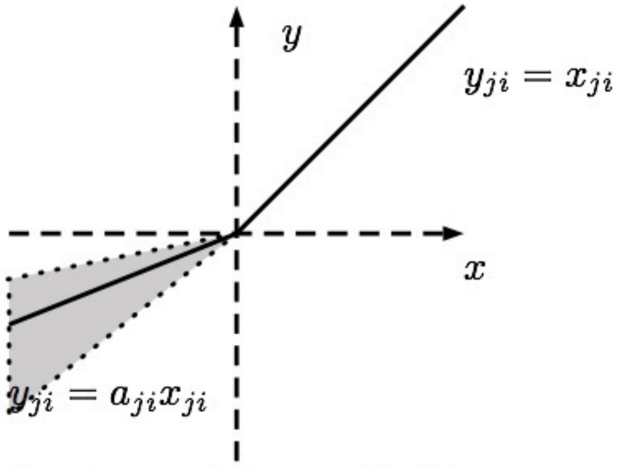
ELU



ReLU



Leaky ReLU/PReLU



Randomized Leaky ReLU

Monitor Your Training Curve

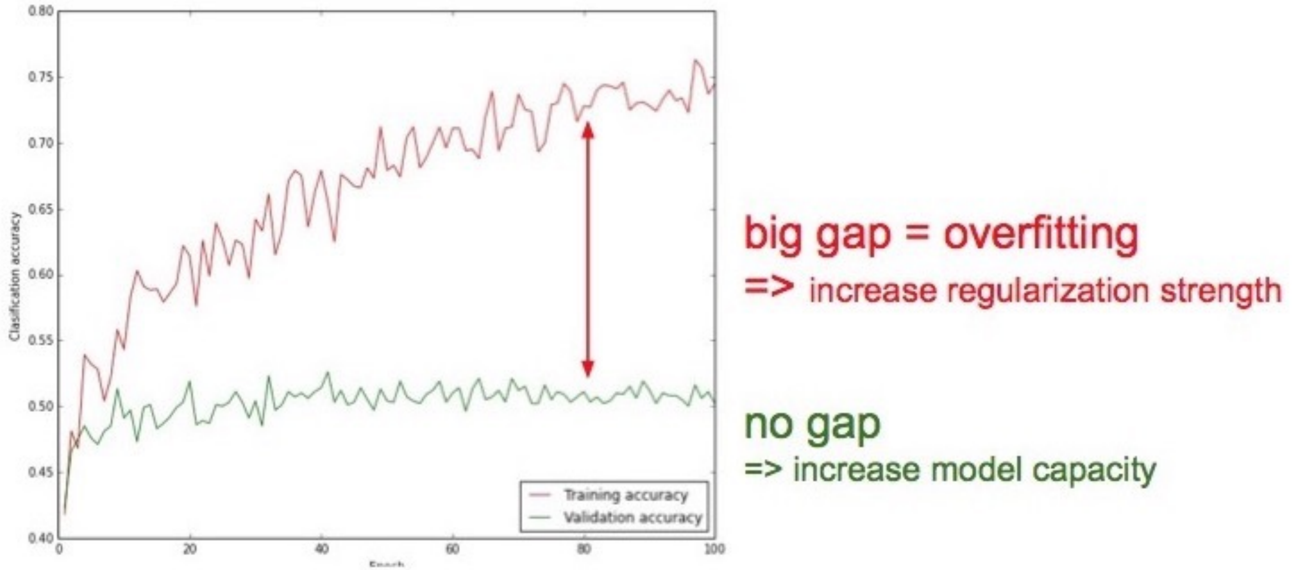


Figure 3

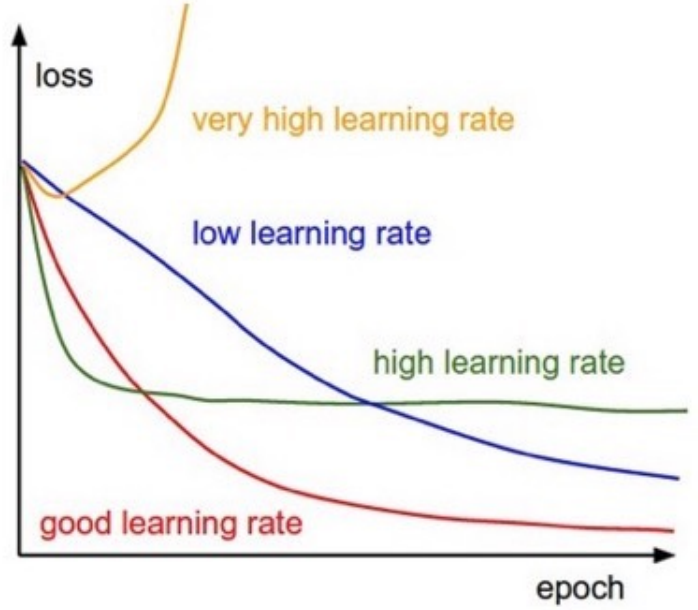


Figure 1

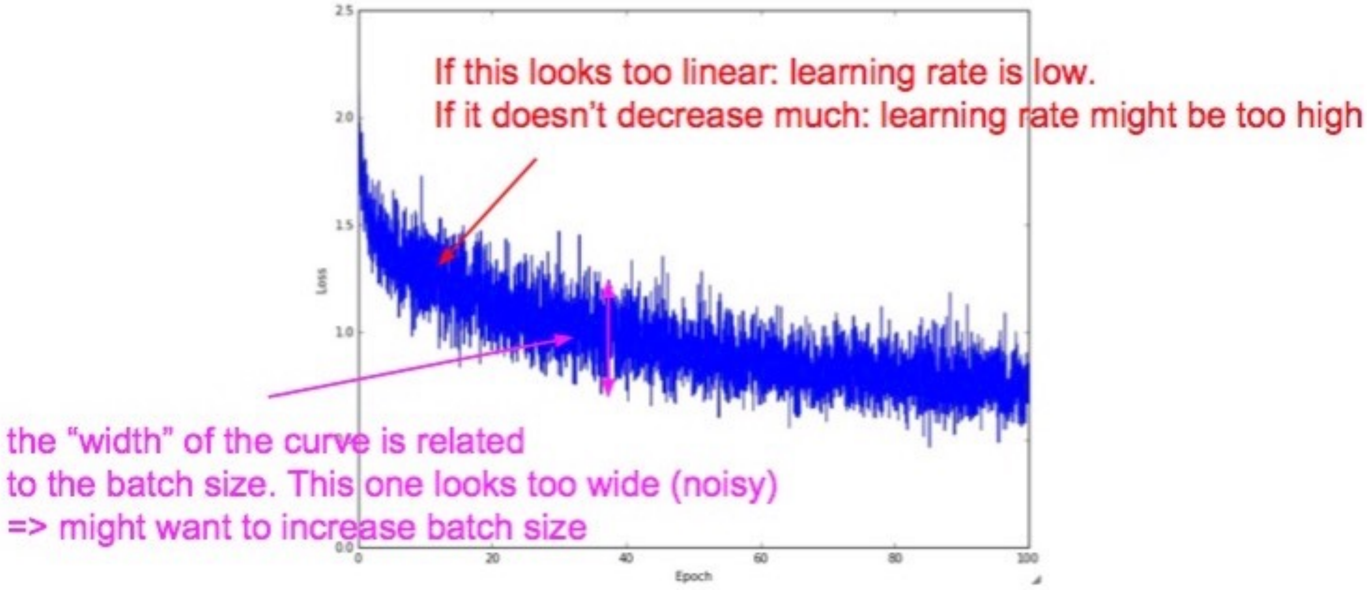


Figure 2

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering