

Introduction to deep learning with neural networks

Jakob Verbeek

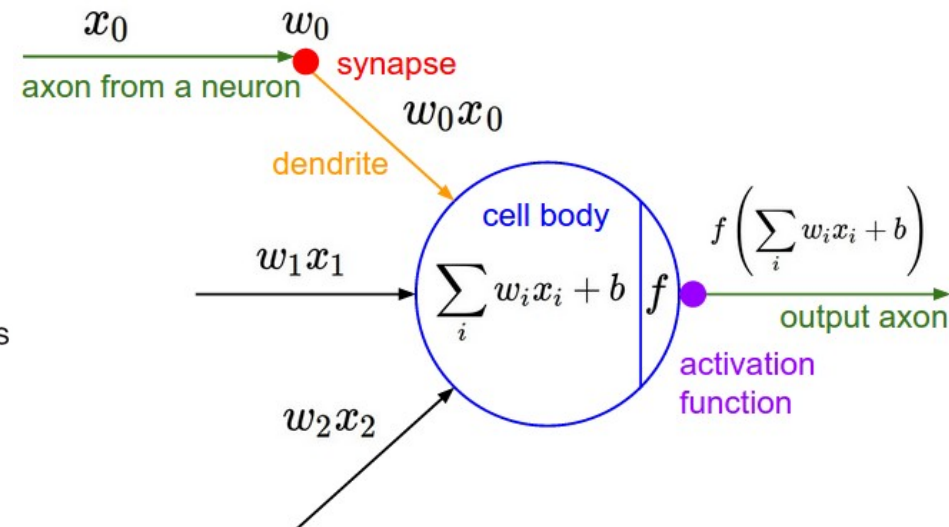
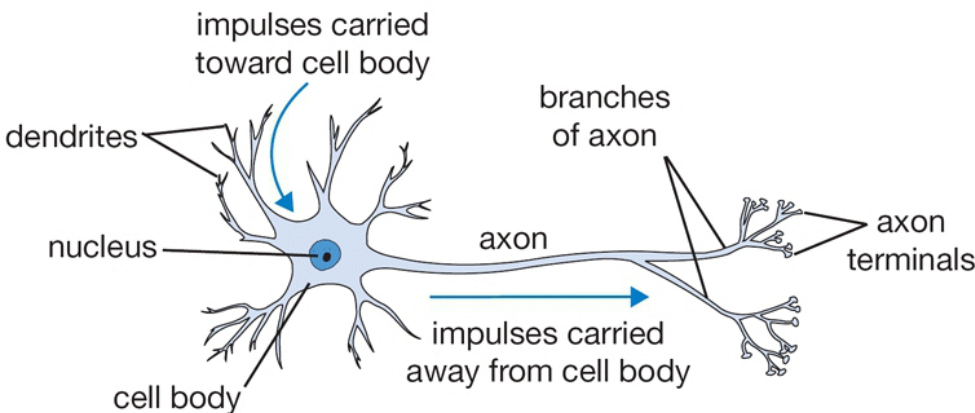
2016 - 2017

Contents

- Feedforward networks
 - ▶ Perceptrons, multi-layer perceptron
 - ▶ Backpropagation
- Convolutional neural networks
 - ▶ Application: object localization
 - ▶ Application: semantic segmentation
- Recurrent neural networks
 - ▶ Gated units
 - ▶ Application: image captioning
- Unsupervised deep learning
 - ▶ Adversarial training
 - ▶ Variational auto-encoders

Biological motivation

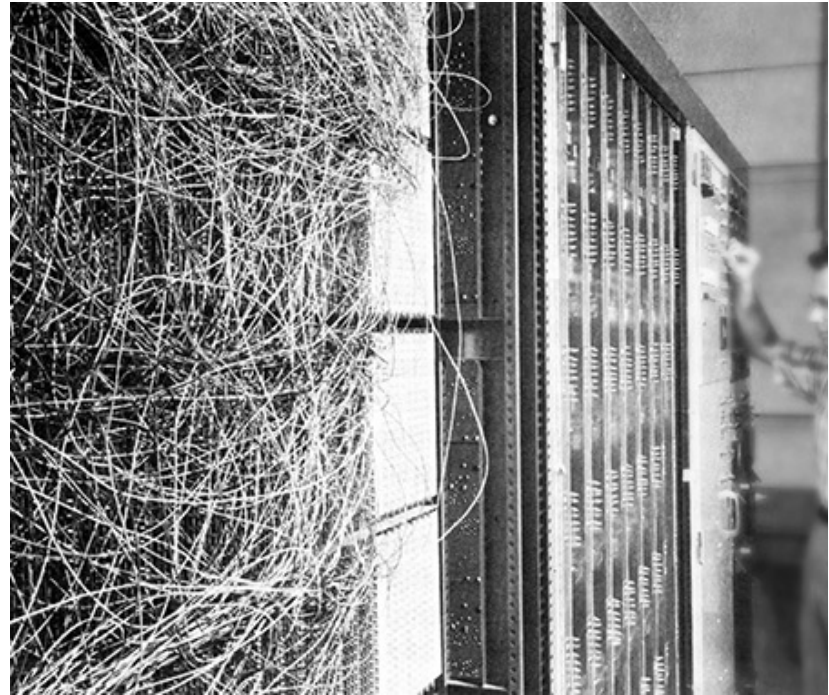
- Neuron is basic computational unit of the brain
 - ▶ about 10^{11} neurons in human brain
- Simplified neuron model as linear threshold unit (McCulloch & Pitts, 1943)
 - ▶ Firing rate of electrical spikes modeled as continuous output quantity
 - ▶ Multiplicative interaction of input and connection strength (weight)
 - ▶ Multiple inputs accumulated in cell activation
 - ▶ Output is non linear function of activation
- Basic component in neural circuits for complex tasks



Rosenblatt's Perceptron



20x20 pixel sensor



Random wiring of associative units

Rosenblatt's Perceptron

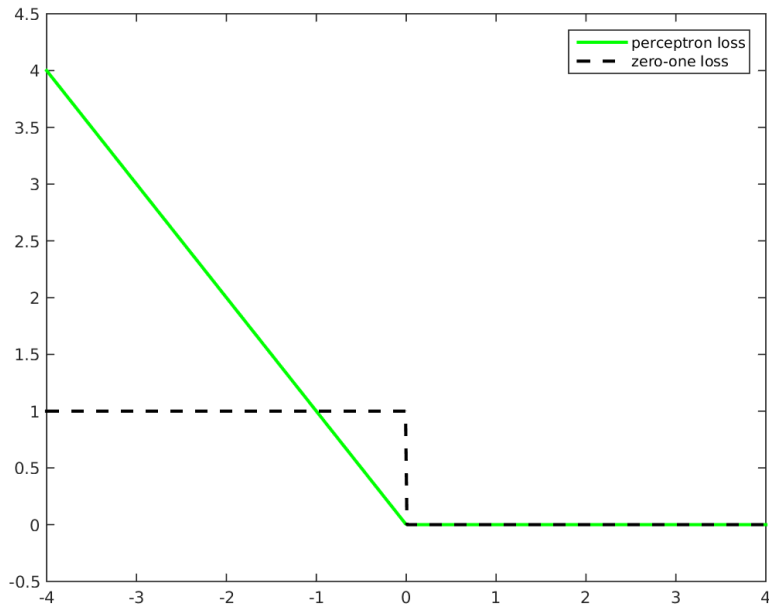
- Objective function linear in score over misclassified patterns $t_i \in \{-1, +1\}$

$$E(w) = -\sum_{t_i \neq \text{sign}(f(x_i))} t_i f(x_i) = \sum_i \max(0, -t_i f(x_i))$$

- Perceptron learning via stochastic gradient descent

$$w^{n+1} = w^n + \eta \times t_i \phi(x_i) \times [t_i f(x_i) < 0]$$

- ▶ Eta is the learning rate



Potentiometers as weights, adjusted by motors during learning

Perceptron convergence theorem

- If a correct solution w^* exists, then the perceptron learning rule will converge to a correct solution in a finite number of iterations for any initial weight vector
- Assume input lives in L2 ball of radius M , and without loss of generality that
 - ▶ w^* has unit L2 norm
 - ▶ Some margin exists for the right solution $y \langle w^*, x \rangle > \delta$
- After a weight update $w' = w + yx$ we have $\langle w^*, w' \rangle = \langle w^*, w \rangle + y \langle w^*, x \rangle > \langle w^*, w \rangle + \delta$
- Moreover, since $y \langle w, x \rangle < 0$ for misclassified sample, we have

$$\begin{aligned} \langle w', w' \rangle &= \langle w, w \rangle + 2y \langle w, x \rangle + \langle x, x \rangle \\ &< \langle w, w \rangle + \langle x, x \rangle \\ &< \langle w, w \rangle + M \end{aligned}$$
- Thus after t updates we have

$$\begin{aligned} \langle w^*, w' \rangle &> \langle w^*, w \rangle + t \delta \\ \langle w', w' \rangle &< \langle w, w \rangle + tM \end{aligned}$$

and therefore $a(t) = \frac{\langle w^*, w(t) \rangle}{\sqrt{\langle w(t), w(t) \rangle}} > \frac{\langle w^*, w \rangle + t \delta}{\sqrt{\langle w, w \rangle + tM}}$ in limit of large t : $a(t) > \frac{\delta}{\sqrt{M}} \sqrt{t}$
- Since $a(t)$ is upper bounded by construction by 1, the nr. of updates t must be limited.
- For start at $w=0$, we have that $t \leq \frac{M}{\delta^2}$

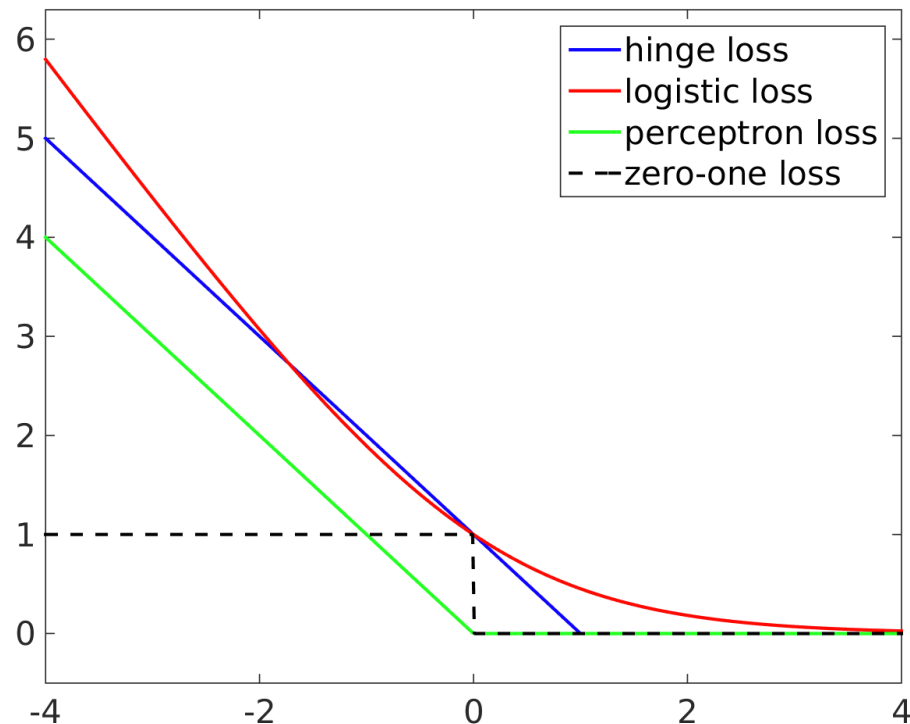
Limitations of the Perceptron

- Perceptron convergence theorem (Rosenblatt, 1962) states that
 - ▶ If training data is linearly separable, then learning algorithm will find a solution in a finite number of iterations
 - ▶ Faster convergence for larger margin (at fixed data scale)
- If training data is linearly separable then the found solution will depend on the initialization and ordering of data in the updates
- If training data is not linearly separable, then the perceptron learning algorithm will not converge
- No direct multi-class extension
- No probabilistic output or confidence on classification

Relation to SVM and logistic regression

- Perceptron loss similar to hinge loss without the notion of margin
 - ▶ Cost function is not a bound on the zero-one loss
- All are either based on linear function or generalized linear function by relying on pre-defined non-linear data transformation

$$f(x) = w^T \phi(x)$$



Kernels to go beyond linear classification

- Representer theorem states that in all these cases optimal weight vector is linear combination of training data

$$w = \sum_i \alpha_i \phi(x_i)$$

$$f(x) = w^T \phi(x) = \sum_i \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

- Kernel trick allows us to compute dot-products between (high-dimensional) embedding of the data

$$k(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$$

- Classification function is linear in data representation given by kernel evaluations over the training data

$$f(x) = \sum_i \alpha_i k(x, x_i) = \alpha^T k(x, \cdot)$$

Limitation of kernels

- Classification based on weighted “similarity” to training samples
 - ▶ Design of kernel based on domain knowledge and experimentation

$$f(x) = \sum_i \alpha_i k(x, x_i) = \alpha^T k(x, .)$$

- ▶ Some kernels are data adaptive, for example the Fisher kernel
 - ▶ Still kernel is designed before and separately from classifier training
-
- Number of free variables grows linearly in the size of the training data
 - ▶ Unless a finite dimensional explicit embedding is available $\phi(x)$
 - ▶ Sometimes kernel PCA is used to obtain such a explicit embedding
-
- Alternatively: fix the number of “basis functions” in advance
 - ▶ Choose a family of non-linear basis functions
 - ▶ Learn the parameters, together with those of linear function

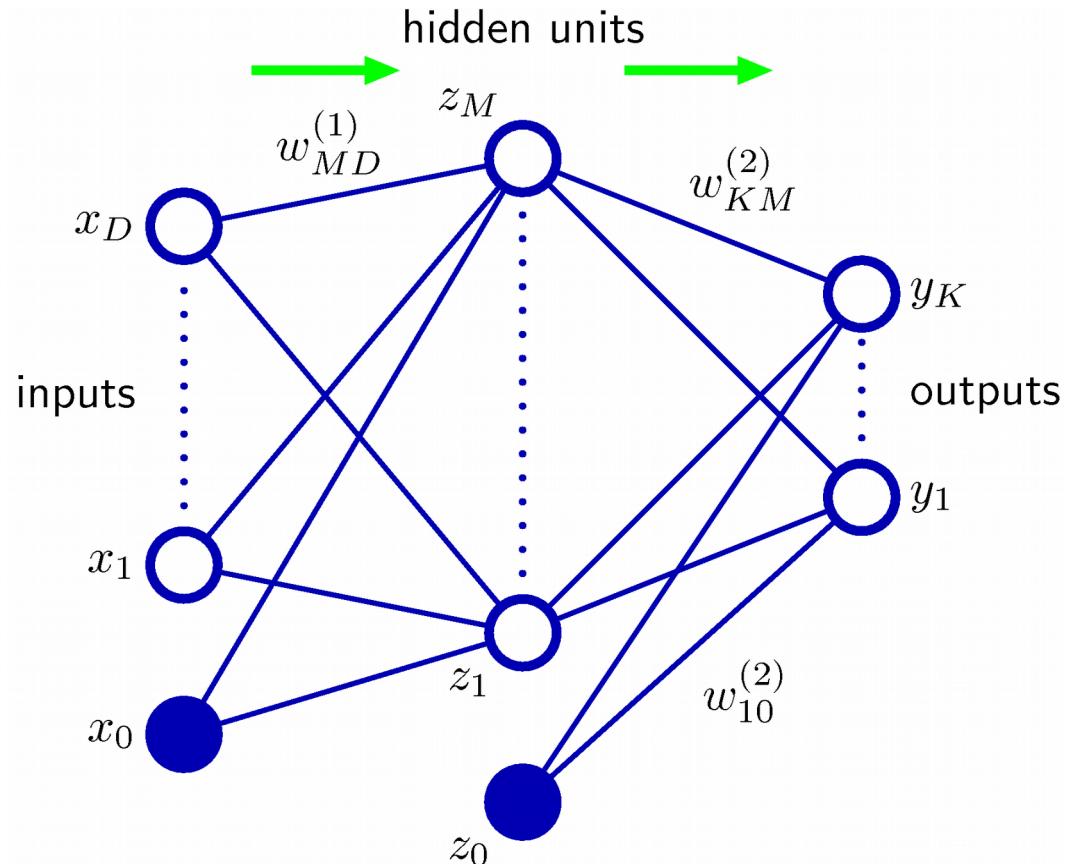
$$f(x) = \sum_i \alpha_i \phi_i(x; \theta_i)$$

Feed-forward neural networks

- Define outputs of one layer as scalar non-linearity of linear function of input
- Known as “multi-layer perceptron” (MLP)
 - ▶ Perceptron has a step non-linearity of linear function (historical)
 - ▶ Other non-linearities are used in practice (see later)

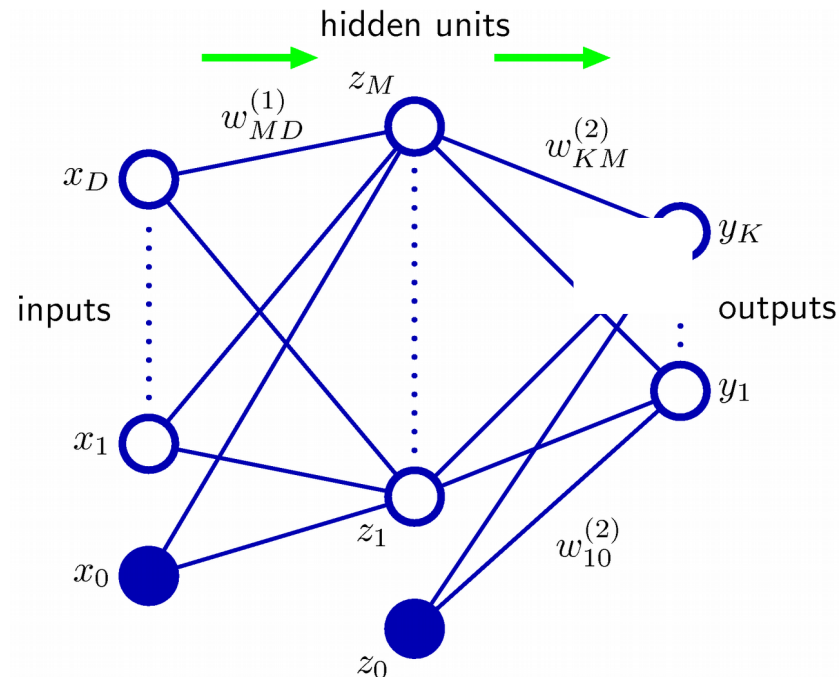
$$z_j = h\left(\sum_i x_i w_{ij}^{(1)}\right)$$

$$y_k = \sigma\left(\sum_j z_j w_{jk}^{(2)}\right)$$



Feed-forward neural networks

- If “hidden layer” activation function is taken to be linear than a single-layer linear model is obtained
- Two-layer networks can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units
 - ▶ Holds for many non-linearities, but not for polynomials



MLP can implement any classifier over binary inputs

- Consider simple case with **binary units**
 - ▶ Inputs and activations are all +1 or -1
 - ▶ Total number of unique input vectors is 2^D
 - ▶ Classification problem into **two classes**
- Use a hidden unit for each positive sample x_m

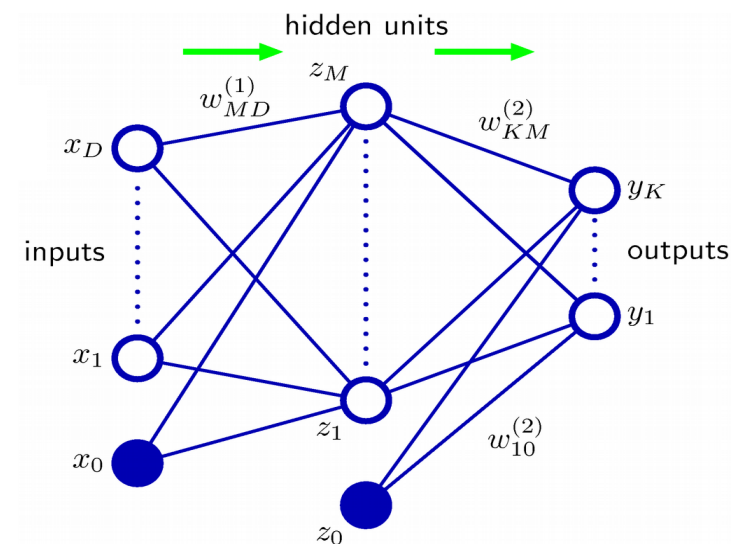
$$z_m = \text{sign}\left(\sum_{i=1}^D w_{mi} x_i - D + 1\right)$$

$$w_{mi} = x_{mi}$$

- ▶ Activation is +1 if and only if input is x_m
- Let output implement an “or” over hidden units

$$y = \text{sign}\left(\sum_{m=1}^M z_m + M - 1\right)$$

- Problem: may need exponential number of hidden units



Feed-forward neural networks

- Architecture can be generalized
 - ▶ More than two layers of computation
 - ▶ Skip-connections from previous layers
- Feed-forward nets are restricted to directed acyclic graphs of connections
 - ▶ Ensures that output can be computed from the input in a single feed-forward pass from the input to the output
- Main issues:
 - ▶ Designing network architecture
 - Nr nodes, layers, non-linearities, etc
 - ▶ Learning the network parameters
 - Non-convex optimization
 - ▶ Sufficient training data
 - Data augmentation, synthesis

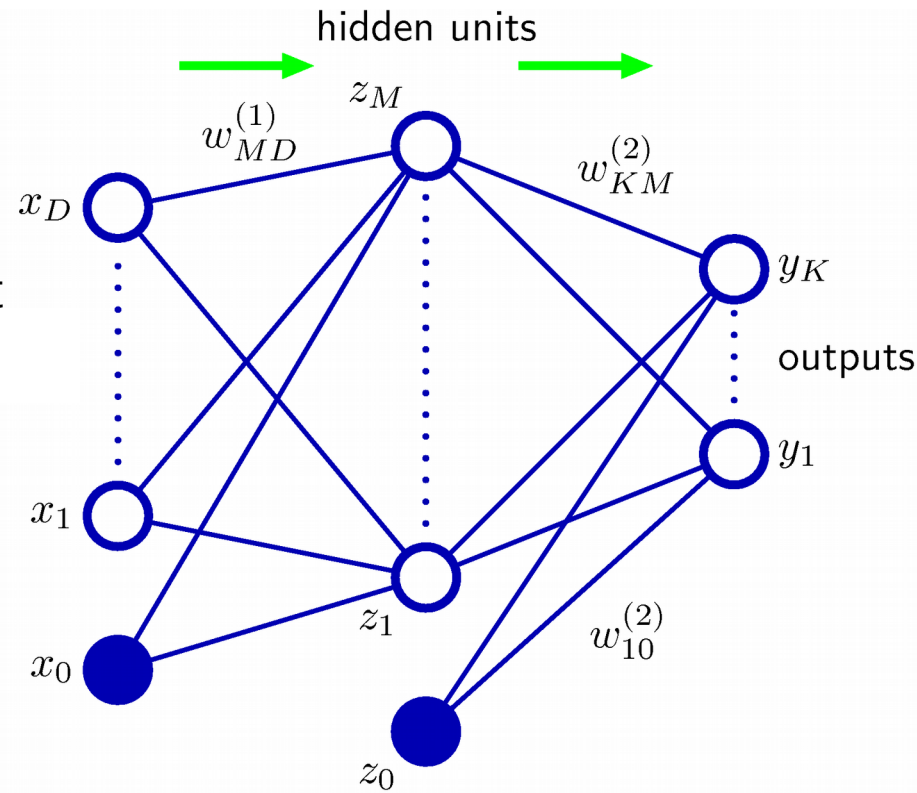


An example: multi-class classification

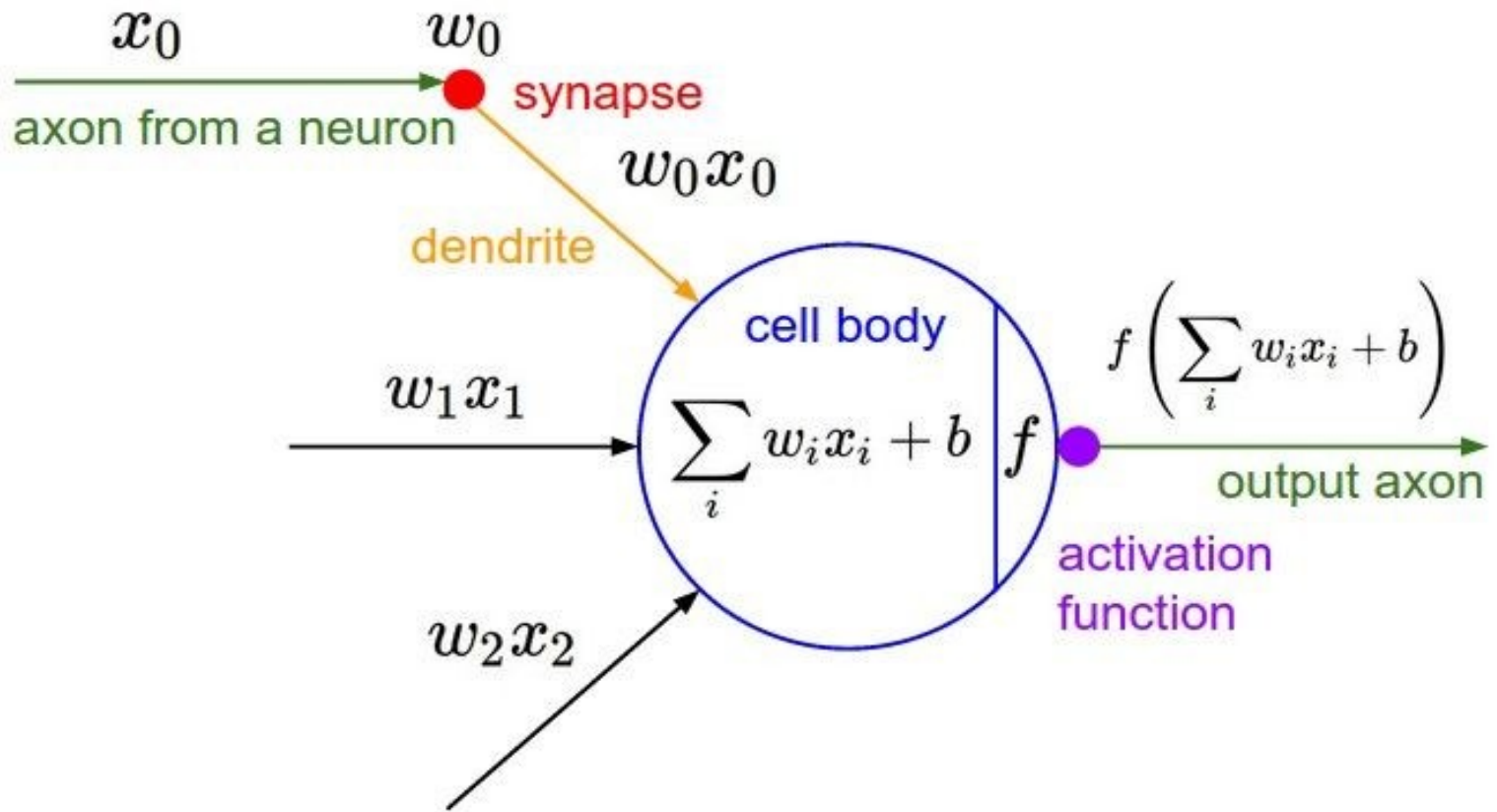
- One output score for each target class
- Multi-class logistic regression loss
 - ▶ Define probability of classes by softmax over scores
 - ▶ Maximize log-probability of correct class
- Precisely as before, but we are now learning the data representation concurrently with the linear classifier

$$p(y=c|x) = \frac{\exp y_c}{\sum_k \exp y_k}$$

- Representation learning in discriminative and coherent manner
- Fisher kernel also data adaptive but not discriminative and task dependent
- More generally, we can choose a loss function for the problem of interest and optimize all network parameters w.r.t. this objective (regression, metric learning, ...)



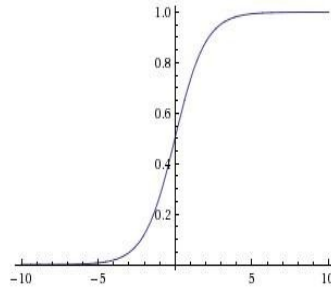
Activation functions



Activation functions

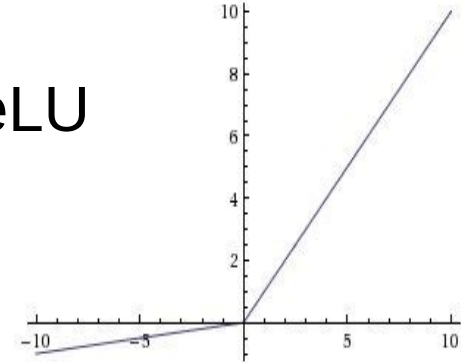
Sigmoid

$$1/(1+e^{-x})$$

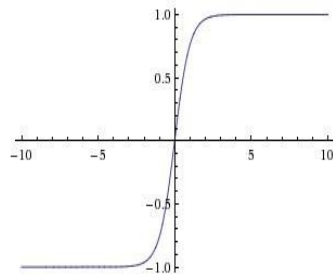


Leaky ReLU

$$\max(\alpha x, x)$$



tanh

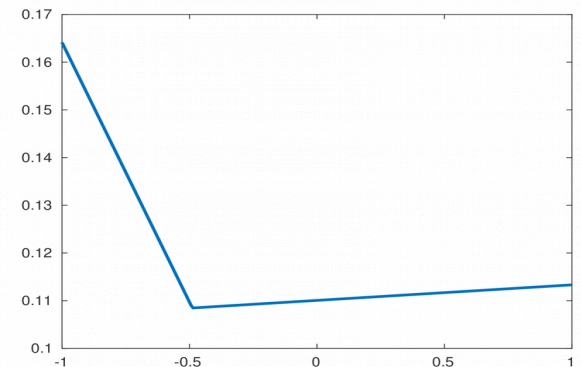
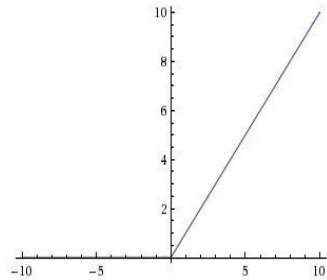


Maxout

$$\max(w_1^T x, w_2^T x)$$

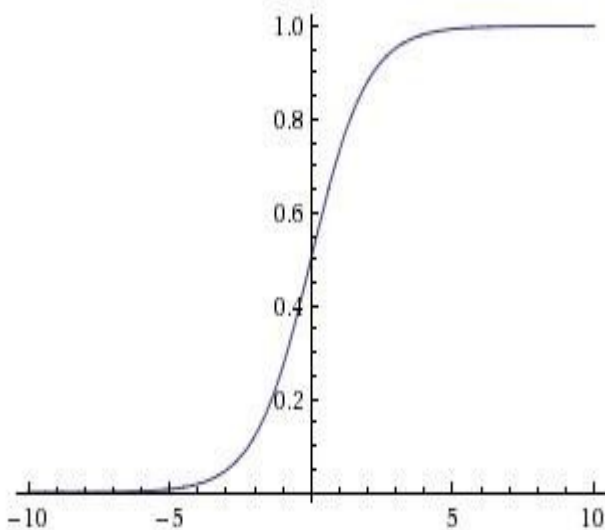
ReLU

$$\max(0, x)$$



Activation Functions

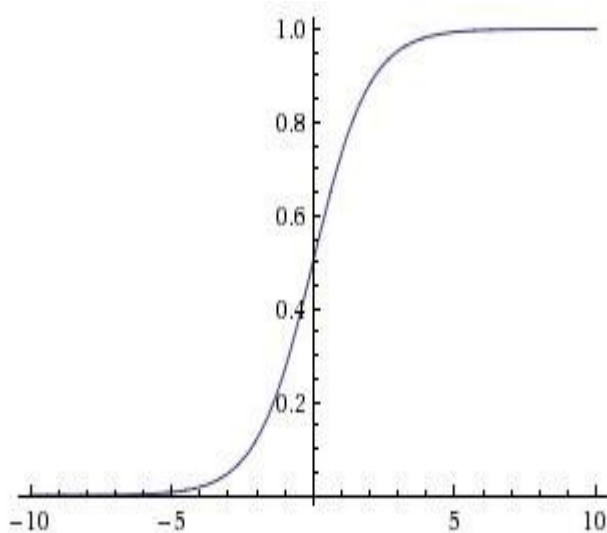
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron



Sigmoid

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

Activation Functions



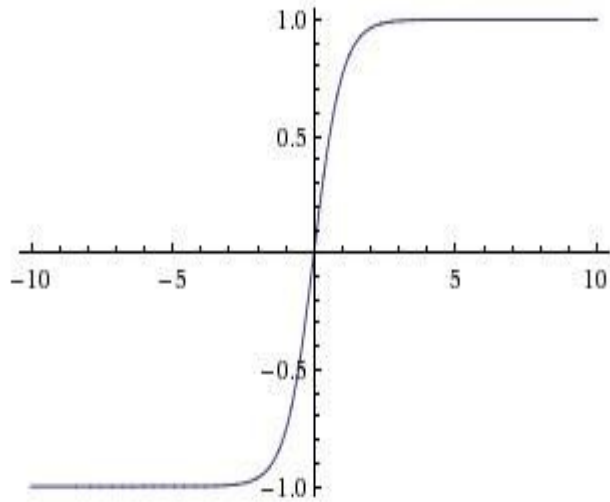
Sigmoid

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

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit compute expensive

Activation Functions



$\tanh(x)$

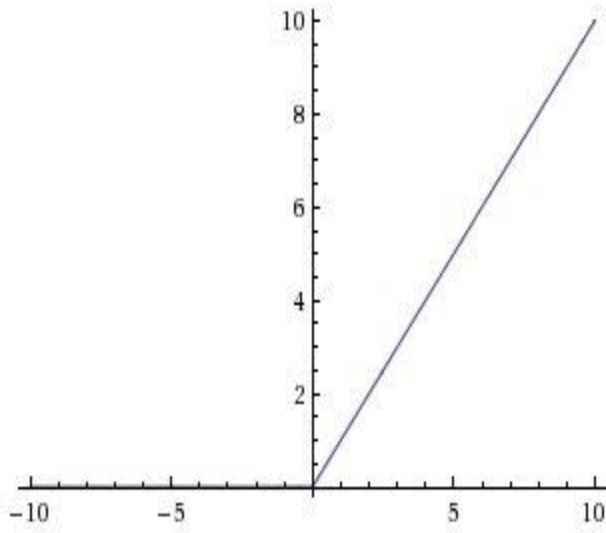
- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions

Computes $f(x) = \max(0, x)$

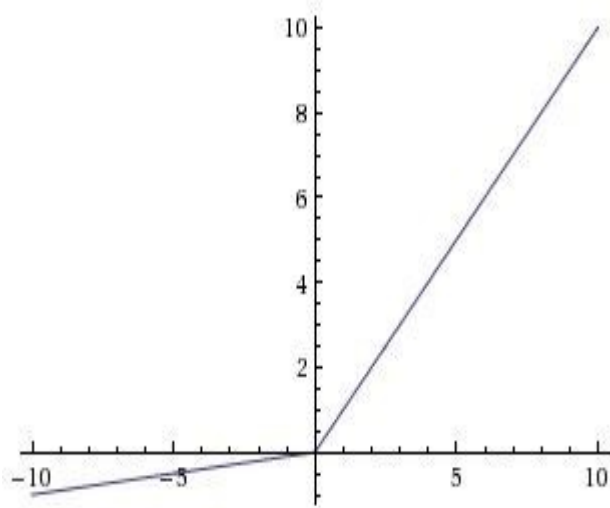
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)



ReLU
(Rectified Linear Unit)

[Nair & Hinton, 2010]

Activation Functions



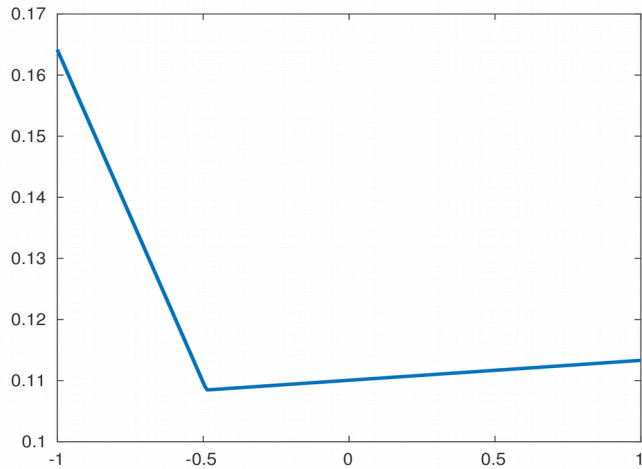
Leaky ReLU

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not “die”.

$$f(x) = \max(0.01x, x)$$

[Mass et al., 2013] [He et al., 2015]

Activation Functions



- Does not saturate
- Computationally efficient
- Will not “die”
- Maxout networks can implement ReLU networks and vice-versa
- More parameters per node

Maxout

$$\max(w_1^T x, w_2^T x)$$

[Goodfellow et al., 2013]

Training feed-forward neural network

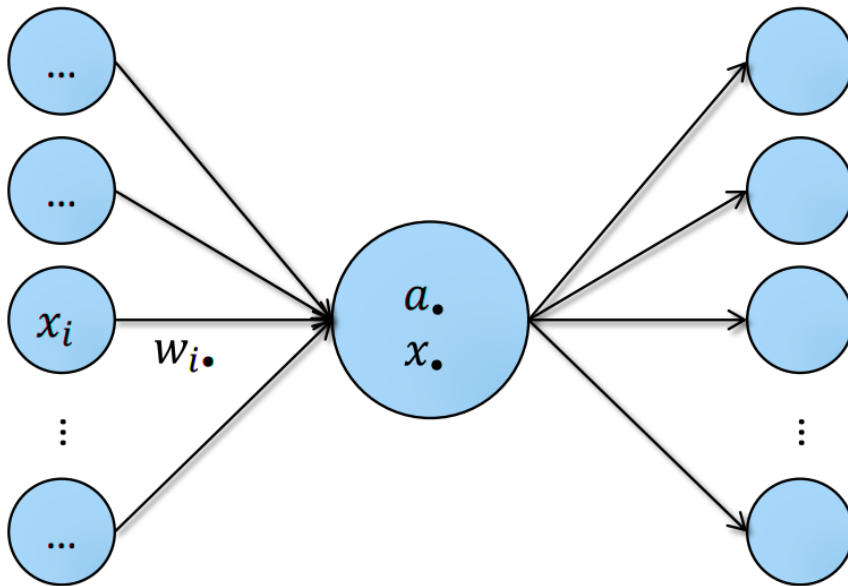
- Non-convex optimization problem in general (or at least in useful cases)
 - ▶ Typically number of weights is (very) large (millions in vision applications)
 - ▶ Seems that many different local minima exist with similar quality

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

- Regularization
 - ▶ L2 regularization: sum of squares of weights
 - ▶ “Drop-out”: deactivate random subset of weights in each iteration
 - Similar to using many networks with less weights (shared among them)
- Training using simple gradient descend techniques
 - ▶ Stochastic gradient descend for large datasets (large N)
 - ▶ Estimate gradient of loss terms by averaging over a relatively small number of samples

Training the network: forward propagation

- Forward propagation from input nodes to output nodes
 - ▶ Accumulate inputs into weighted sum
 - ▶ Apply scalar non-linear activation function f
- Use $\text{Pre}(j)$ to denote all nodes feeding into j



$$a_j = \sum_{i \in \text{Pre}(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

Training the network: backward propagation

- Input aggregation and activation

$$a_j = \sum_{i \in \text{Pre}(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

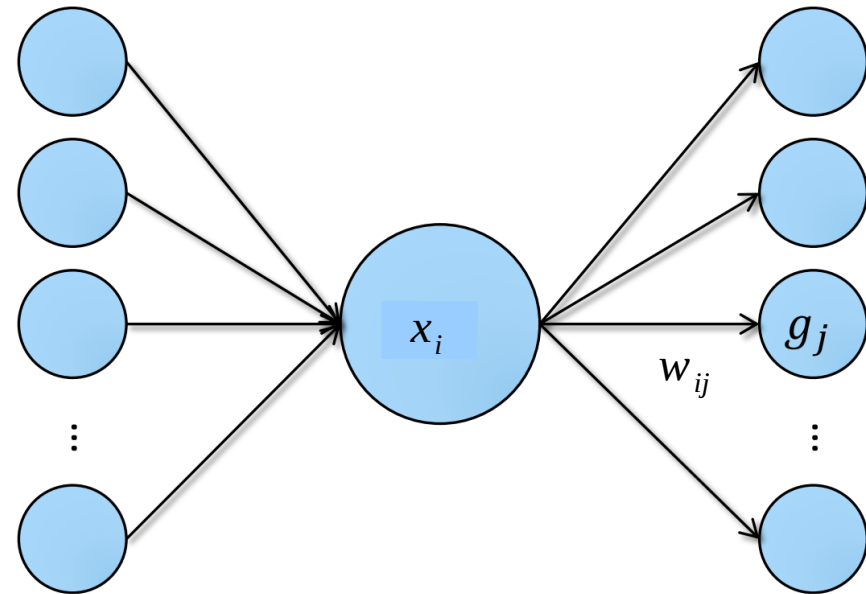
- Partial derivative of loss w.r.t. input

$$g_j = \frac{\partial L}{\partial a_j}$$

- Partial derivative w.r.t. learnable weights

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

- Gradient of weights between two layers given by outer-product of x and g



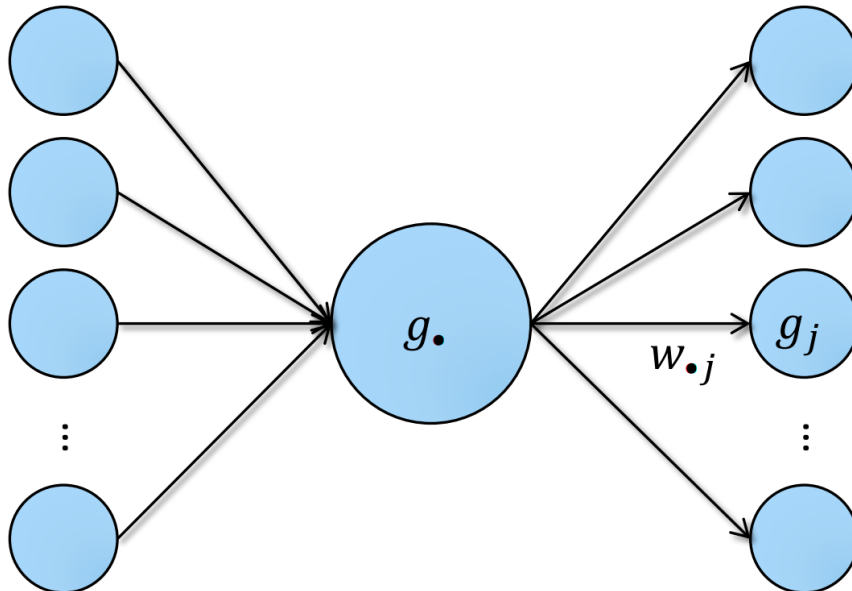
Training the network: backward propagation

- Backward propagation of loss gradient from output nodes to input nodes
 - ▶ Application of chainrule of derivatives
- Accumulate gradients from downstream nodes
 - ▶ Post(i) denotes all nodes that i feeds into
 - ▶ Weights propagate gradient back
- Multiply with derivative of local activation

$$a_j = \sum_{i \in \text{Pre}(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

$$g_i = \frac{\partial L}{\partial a_i}$$



$$\begin{aligned} \frac{\partial L}{\partial x_i} &= \sum_{j \in \text{Post}(i)} \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial x_i} \\ &= \sum_{j \in \text{Post}(i)} g_j w_{ij} \end{aligned}$$

$$\begin{aligned} g_i &= \frac{\partial x_i}{\partial a_i} \frac{\partial L}{\partial x_i} \\ &= f'(a_i) \sum_{j \in \text{Post}(i)} w_{ij} g_j \end{aligned}$$

Limitations recurrent networks

- In a simple RNN the input and previous state always “write” on the state in the same manner
- Limits capacity to capture long-range dependencies as state cannot be “shielded” from updates for long term storage (Hochreiter 91, Bengio '94)
- Input-dependent “gates” can selectively block/pass inputs to the state

Training the network: forward and backward propagation

- Special case for Rectified Linear Unit (ReLU) activations

$$f(a) = \max(0, a)$$

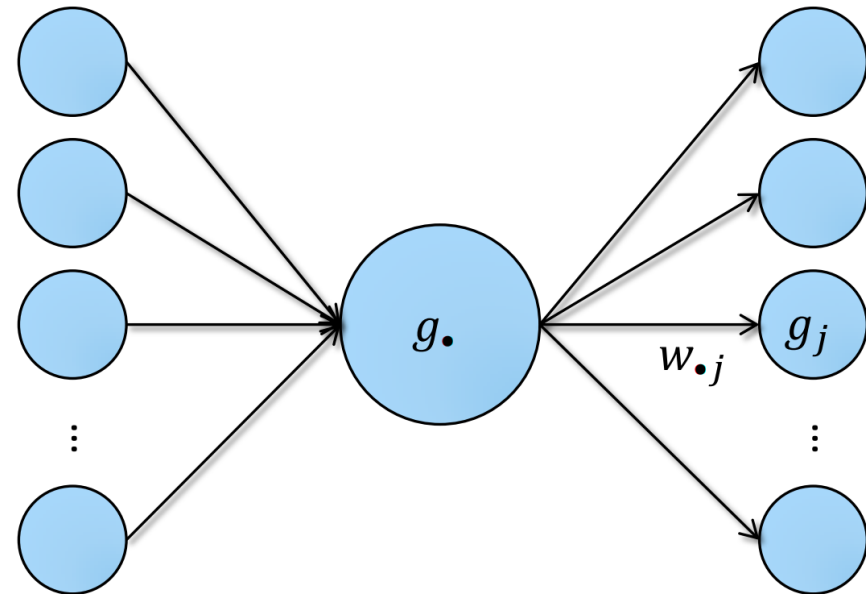
- Sub-gradient is step function

$$f'(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- Sum gradients from downstream nodes

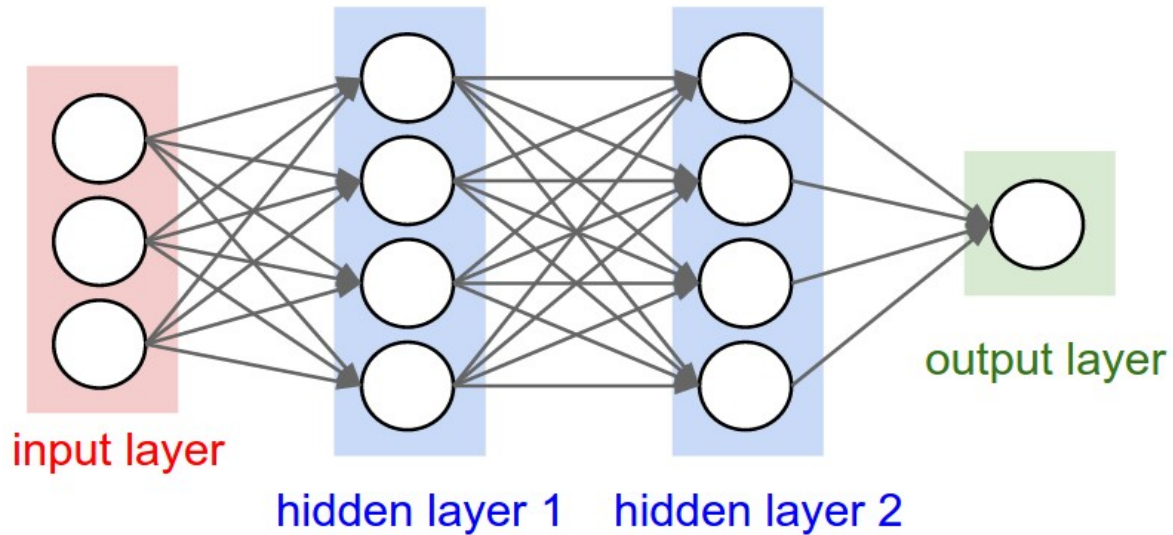
$$g_i = \begin{cases} 0 & \text{if } a_i \leq 0 \\ \sum_{j \in \text{Post}(i)} w_{ij} g_j & \text{otherwise} \end{cases}$$

- ▶ Set to zero if in ReLU zero-regime
- ▶ Compute sum only for active units
- Note how gradient on incoming weights is “killed” by inactive units
 - ▶ Generates tendency for those units to remain inactive



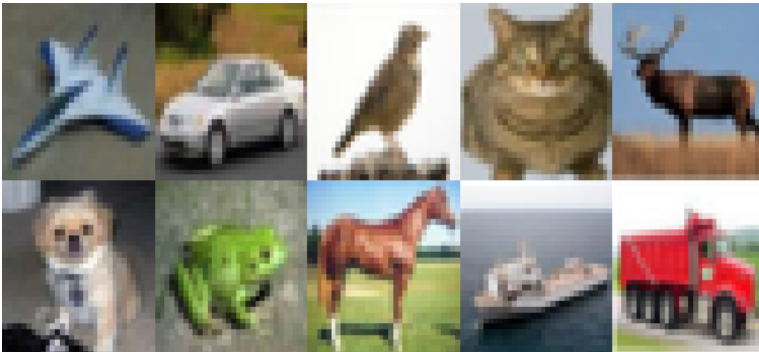
$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

Neural Networks



How to represent the image at the network input?

Input example : an image

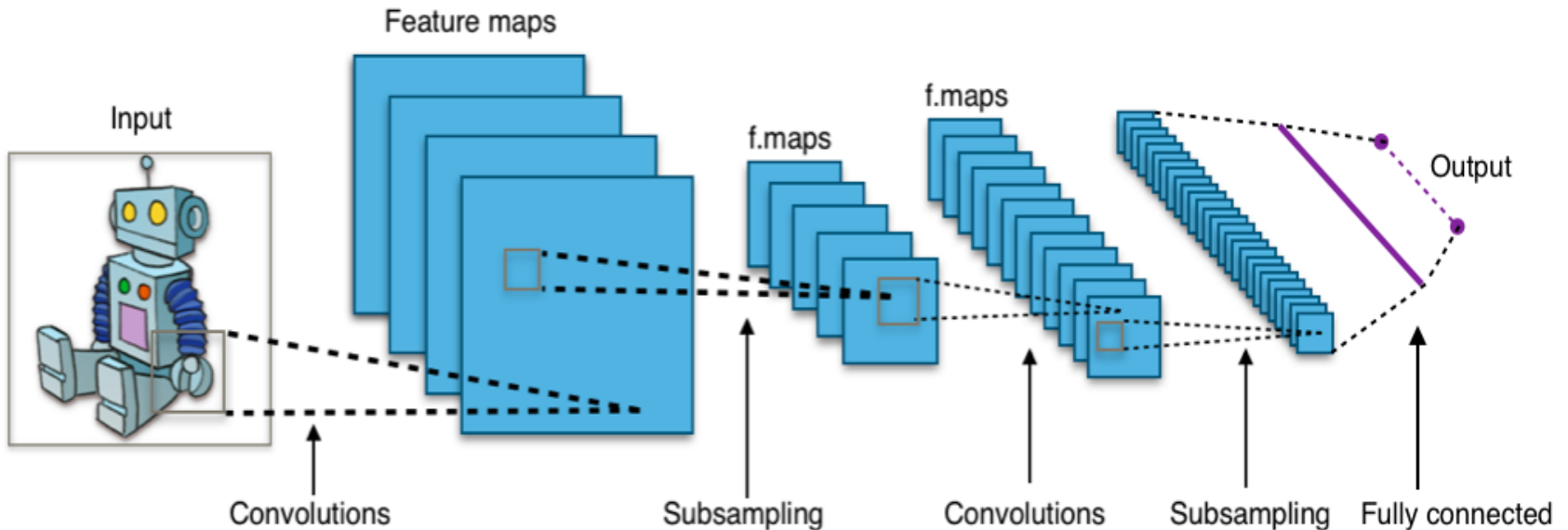


Output example : one class

airplane dog
automobile frog
bird horse
cat ship
deer truck

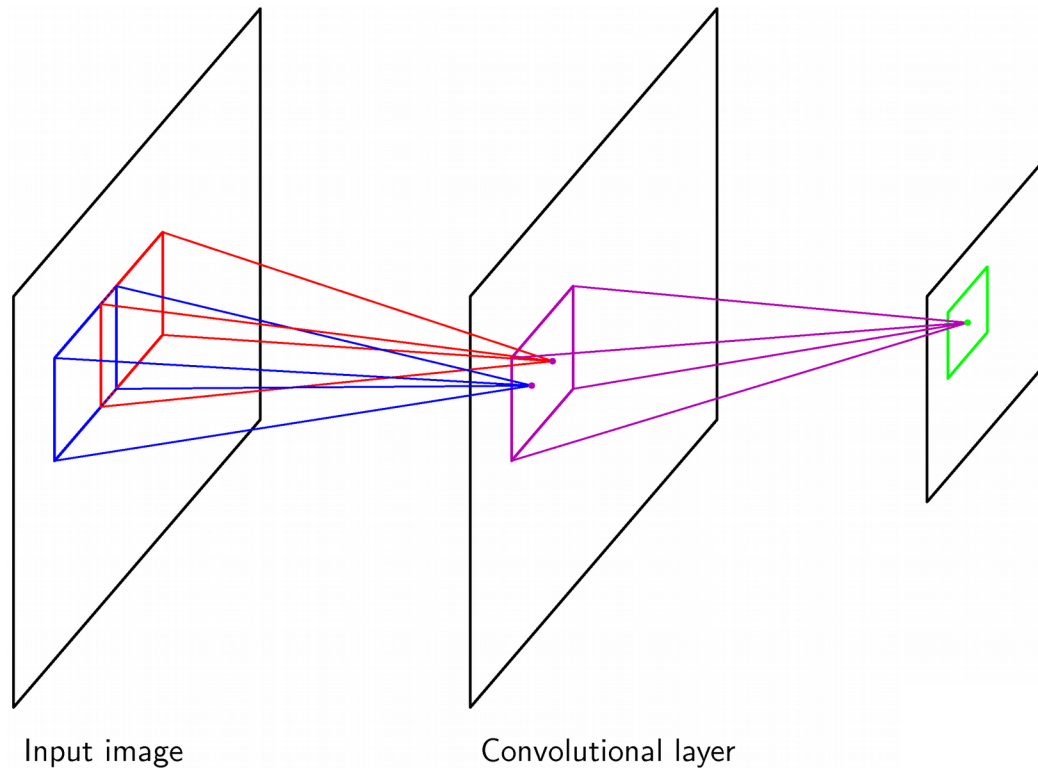
Convolutional neural networks

- A convolutional neural network is a feedforward network where
 - ▶ Hidden units are organized into images or “response maps”
 - ▶ Linear mapping from layer to layer is replaced by convolution



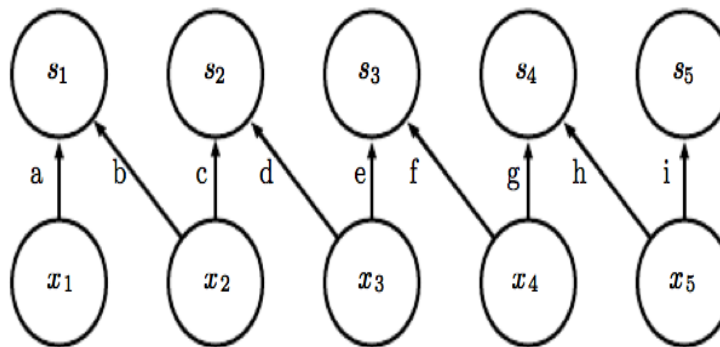
Convolutional neural networks

- Local connections: motivation from findings in early vision
 - ▶ Simple cells detect local features
 - ▶ Complex cells pool simple cells in retinotopic region
- Convolutions: motivated by translation invariance
 - ▶ Same processing should be useful in different image regions

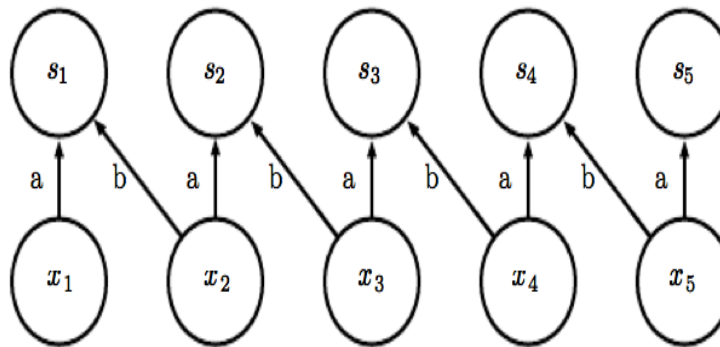


Local connectivity

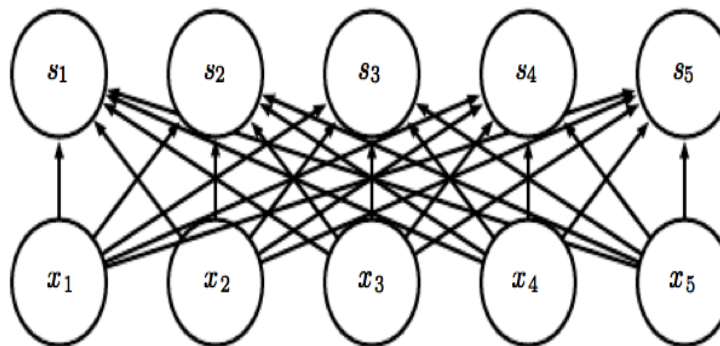
Locally connected layer



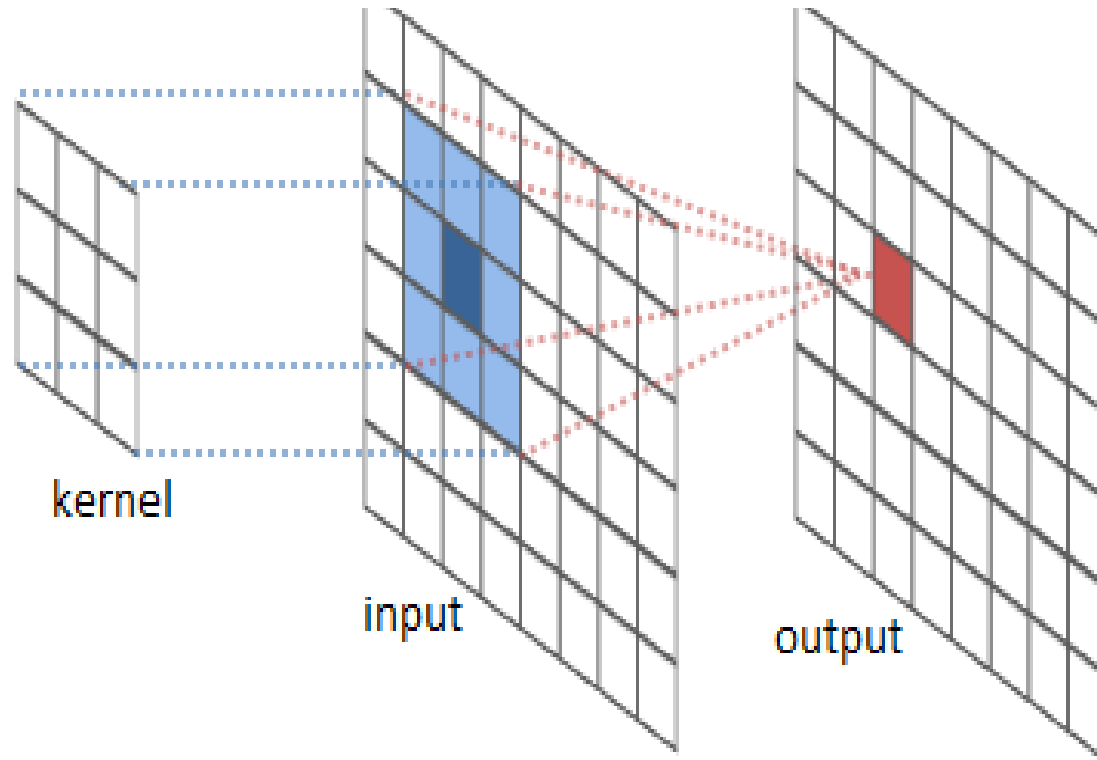
Convolutional layer



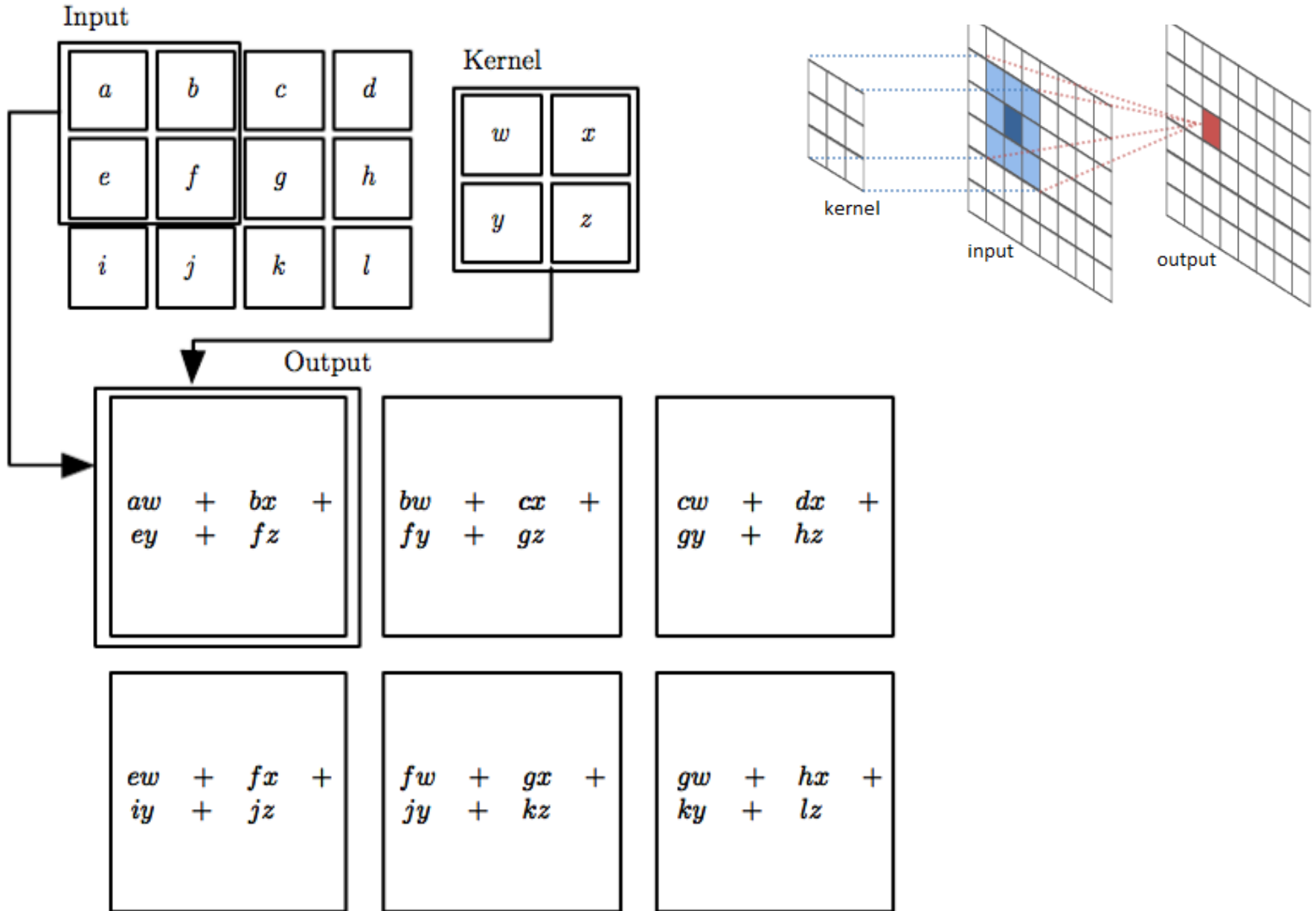
Fully connected layer



The convolution operation

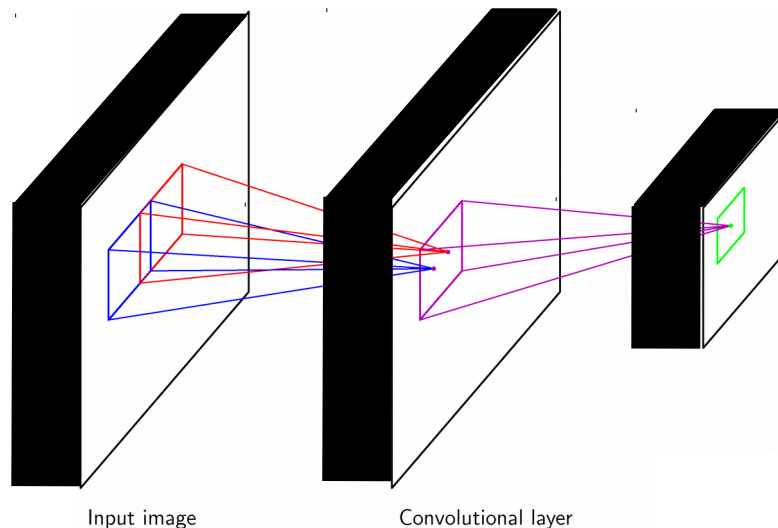


The convolution operation



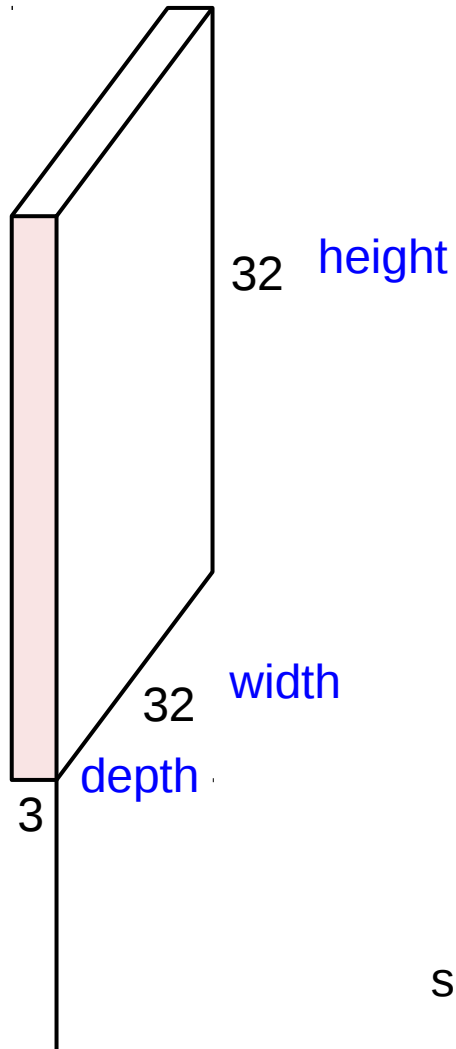
Convolutional neural networks

- Hidden units form another “image” or “response map”
 - ▶ Result of convolution: translation invariant linear function of local inputs
 - ▶ Followed by non-linearity
- Different convolutions can be computed “in parallel”
 - ▶ Gives a “stack” of response maps
 - ▶ Similarly, convolutional filters “read” across different maps
 - ▶ Input may also be multi-channel, e.g. RGB image
- Sharing of weights across hidden units
 - ▶ Number of parameters decoupled from input and representation size



Convolution Layer

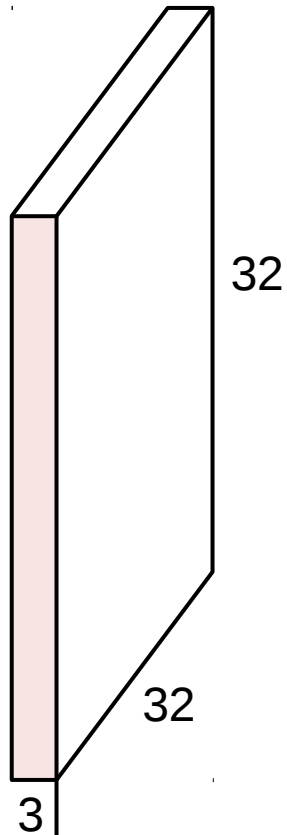
32x32x3 image



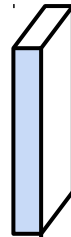
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image



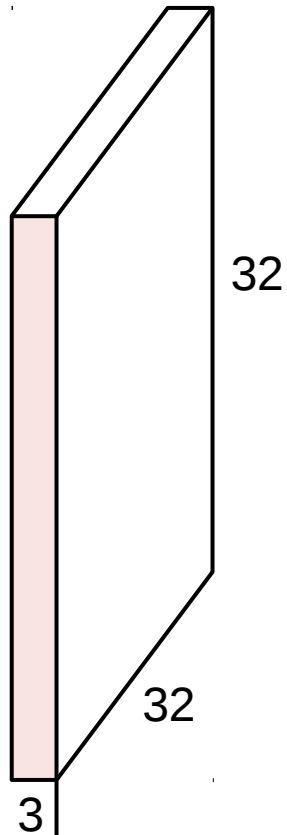
5x5x3 filter



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

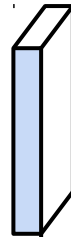
Convolution Layer

32x32x3 image



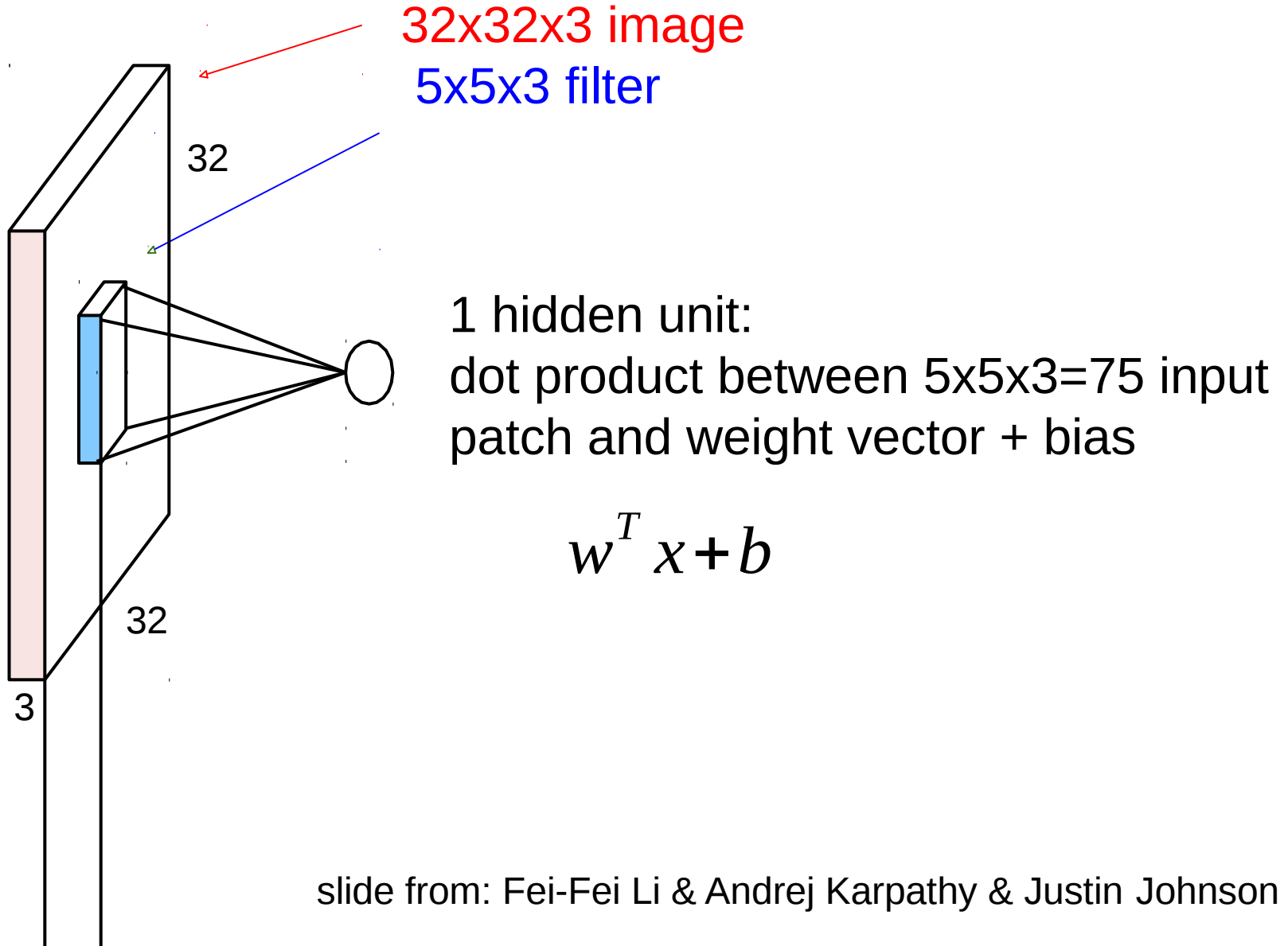
Filters always extend the full depth of the input volume

5x5x3 filter

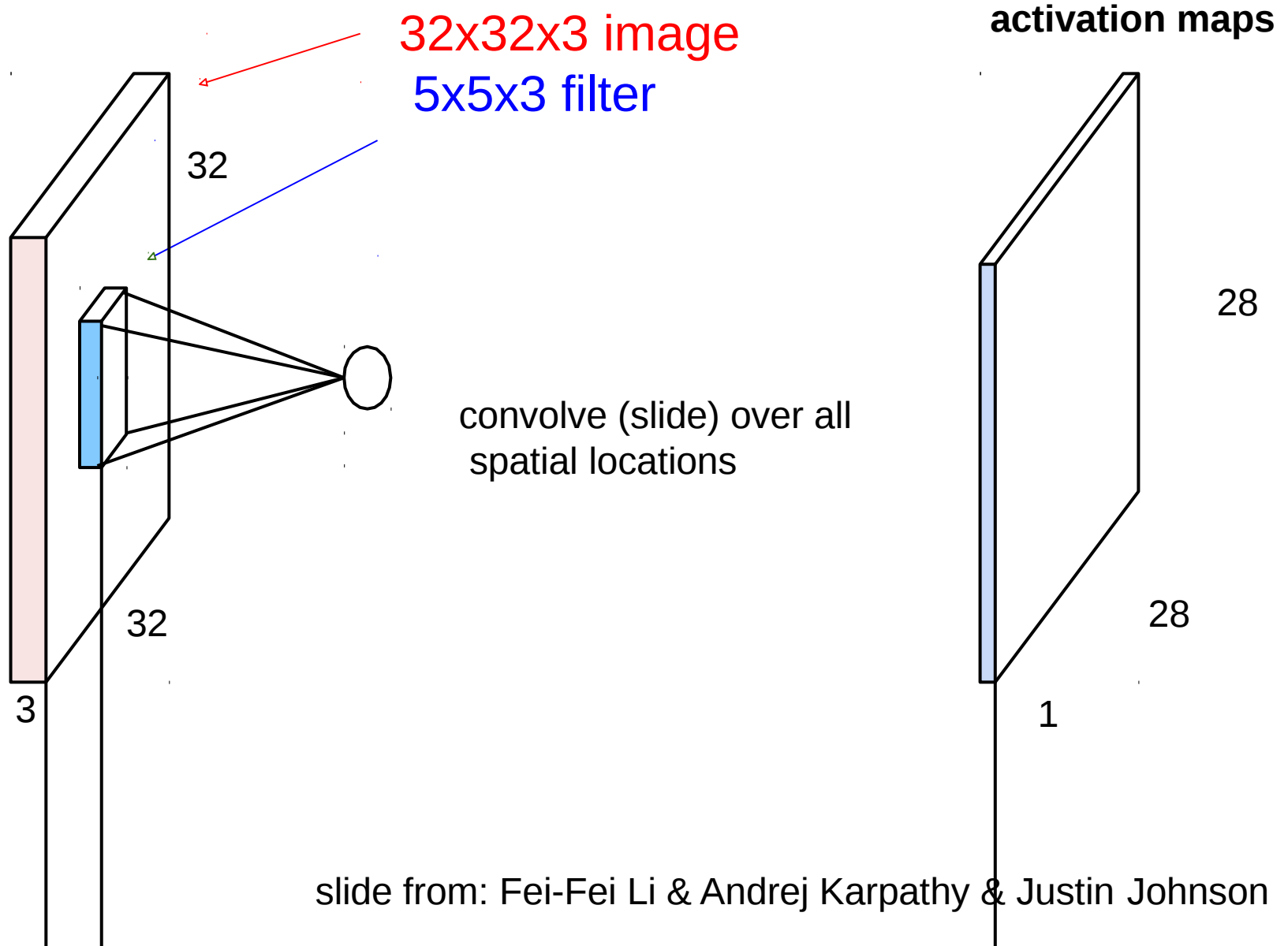


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

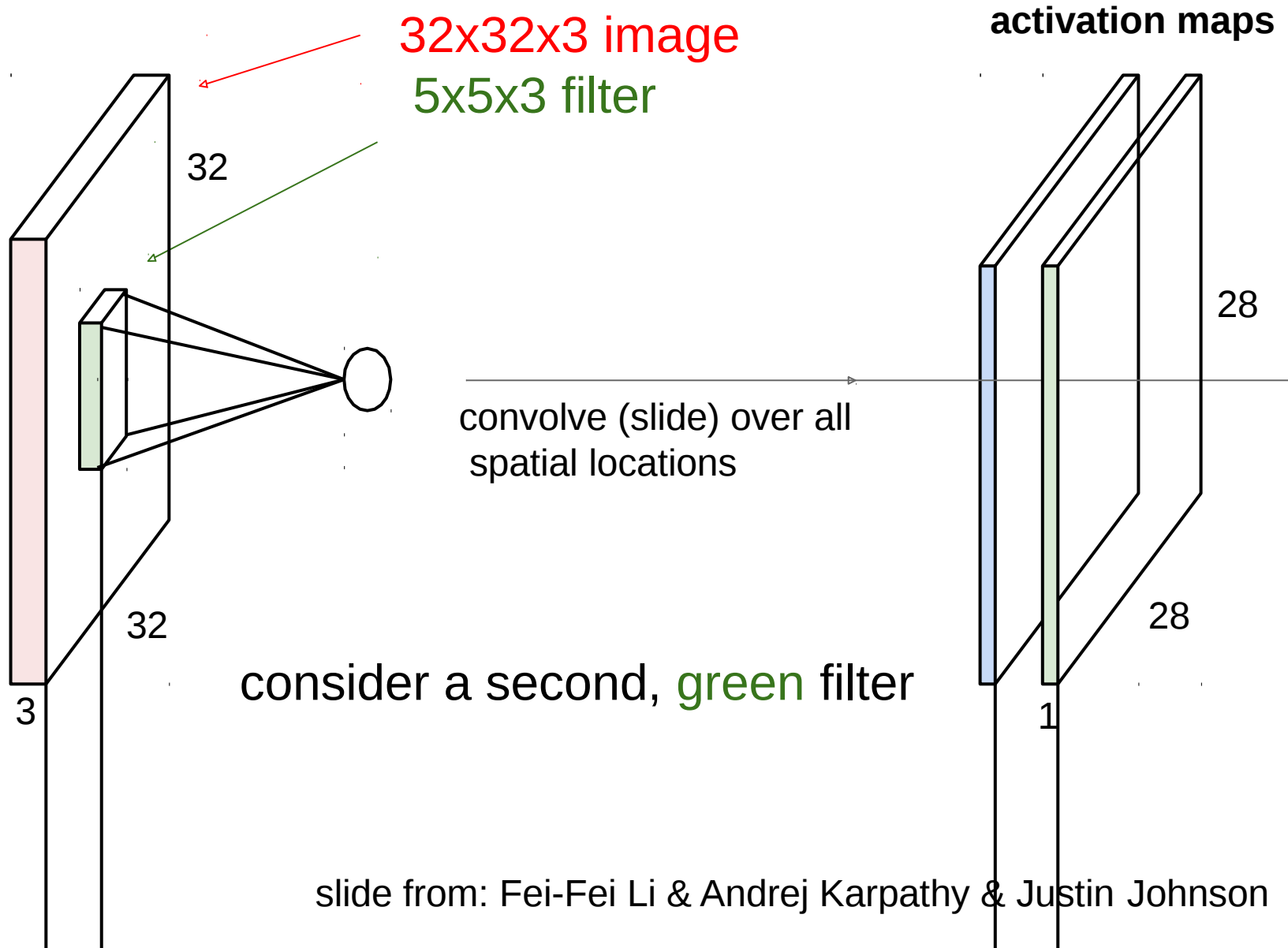


Convolution Layer

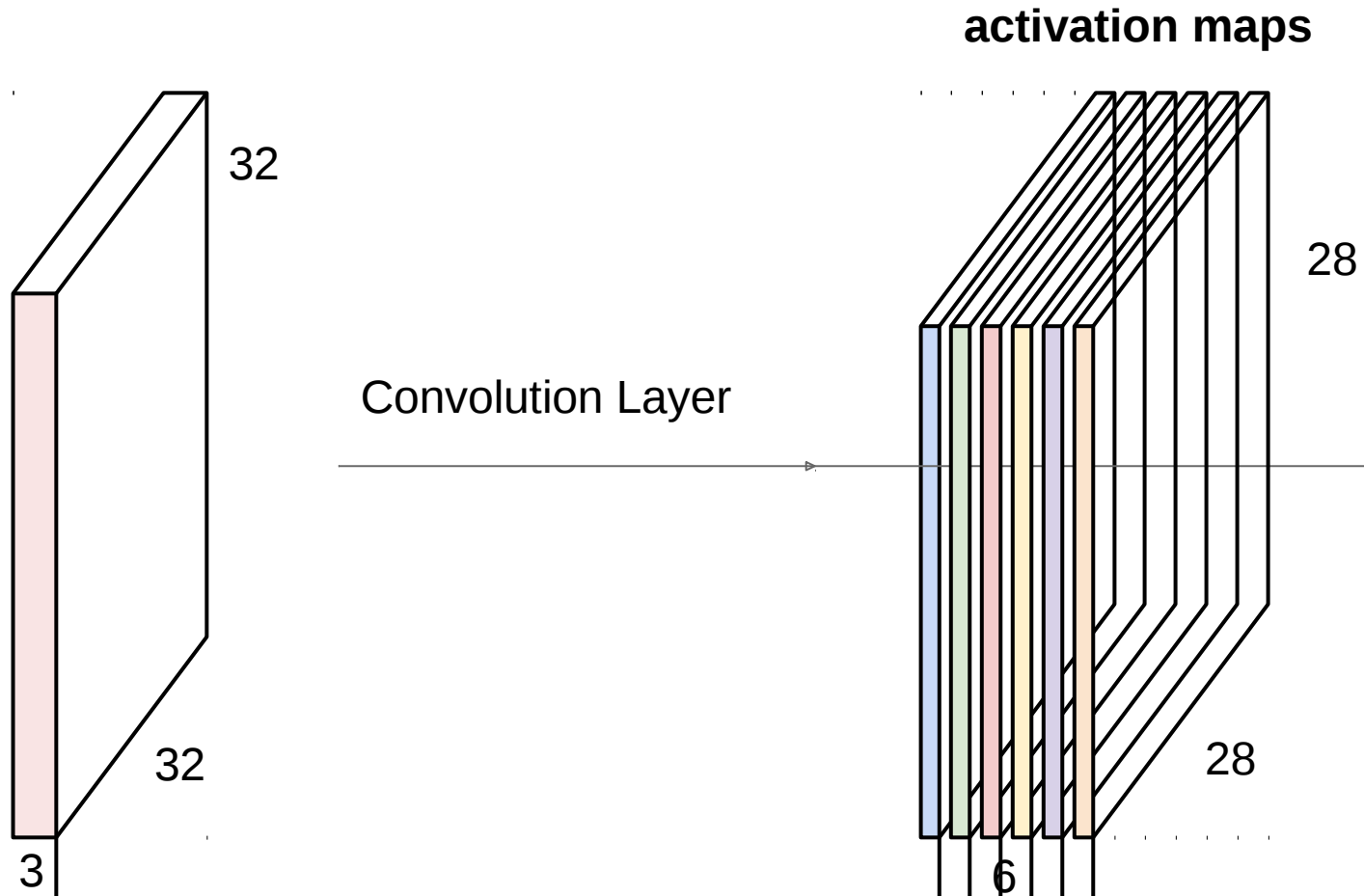


slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

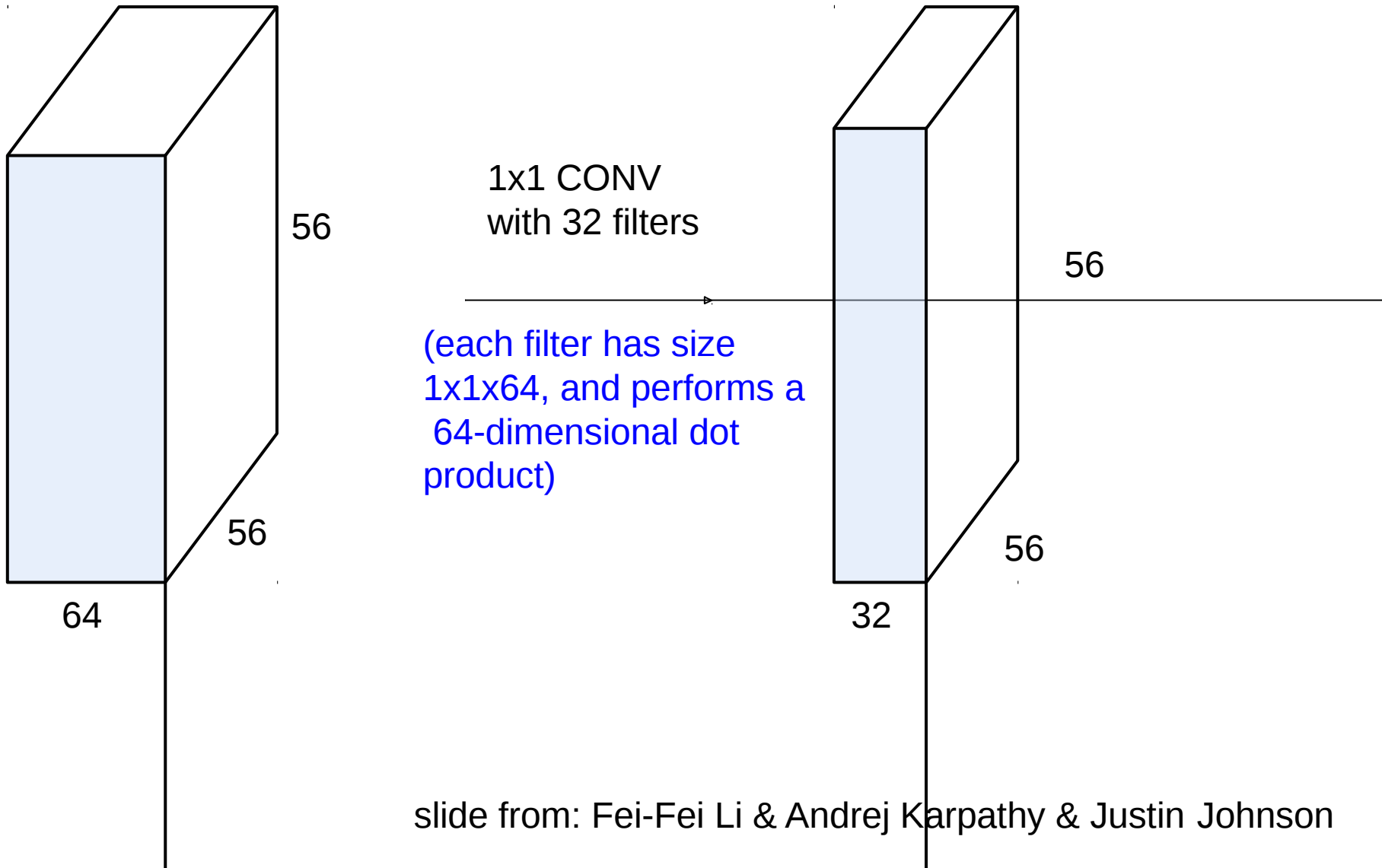


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

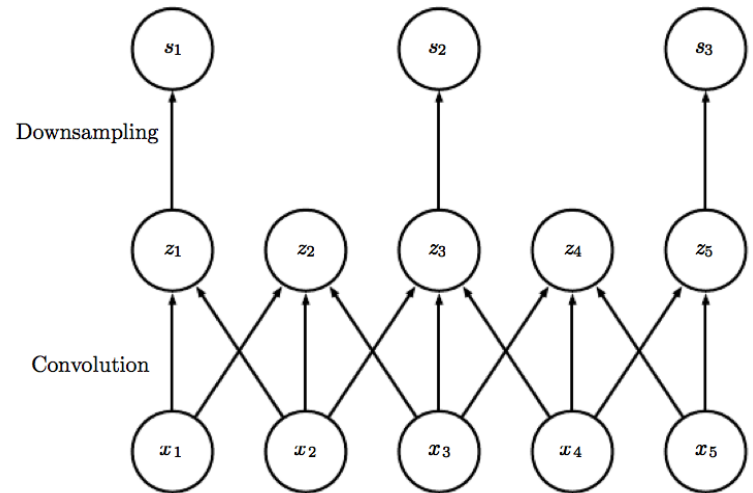
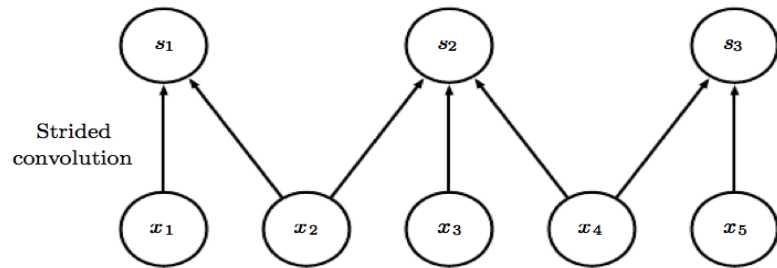


We stack these up to get a "new image" of size 28x28x6!

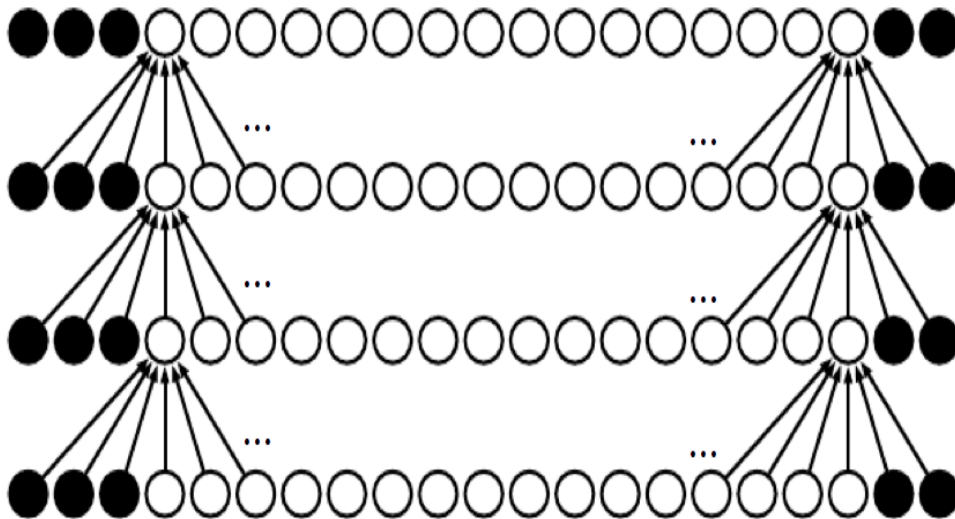
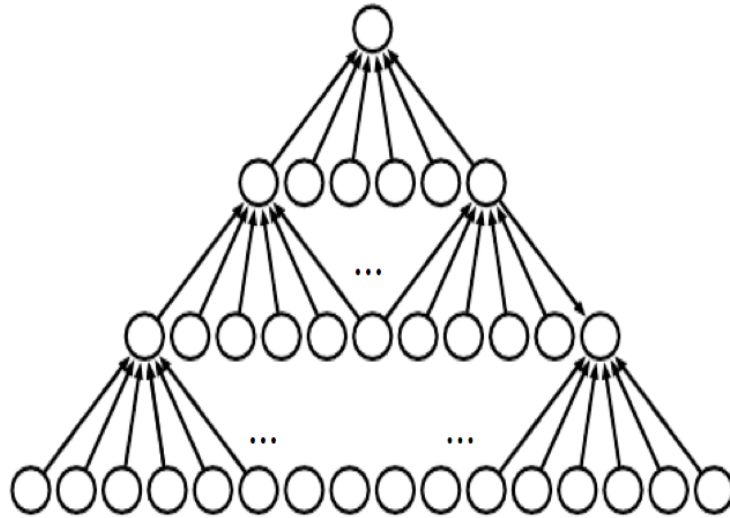
Convolution with 1x1 filters makes perfect sense



Stride



(Zero)-Padding

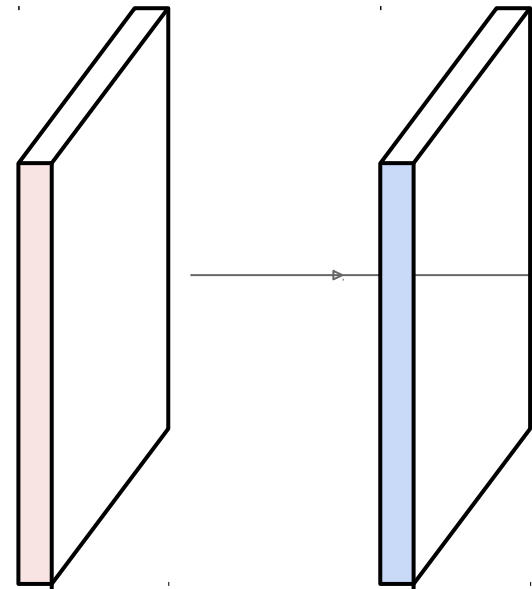


Example:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Example:

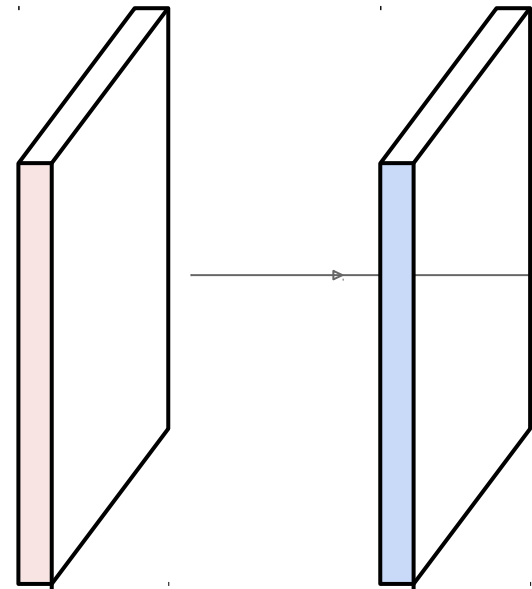
Input volume: **32x32x3**

10 **5x5** filters with stride **1**, pad **2**

Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially, so

32x32x10

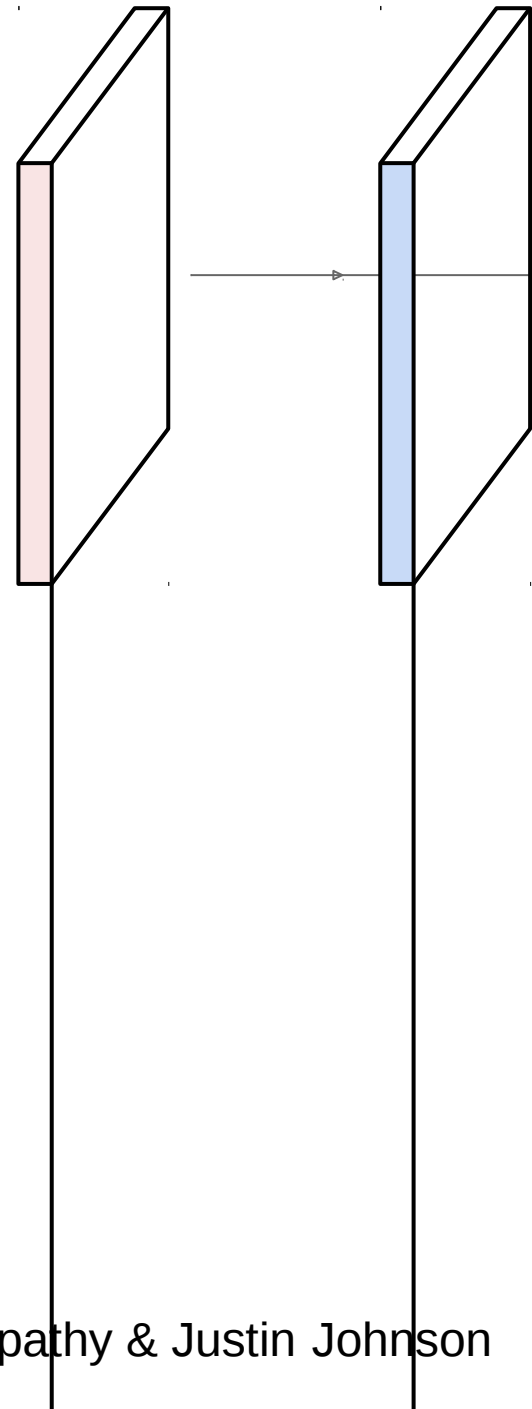


Example:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?



Example:

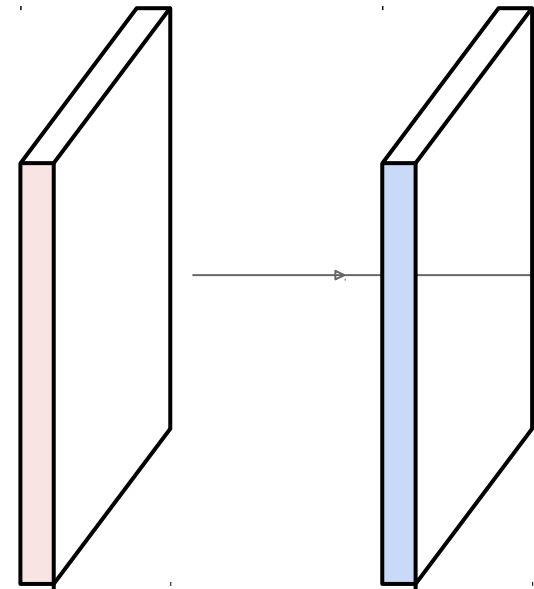
Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2

Number of parameters in this layer?

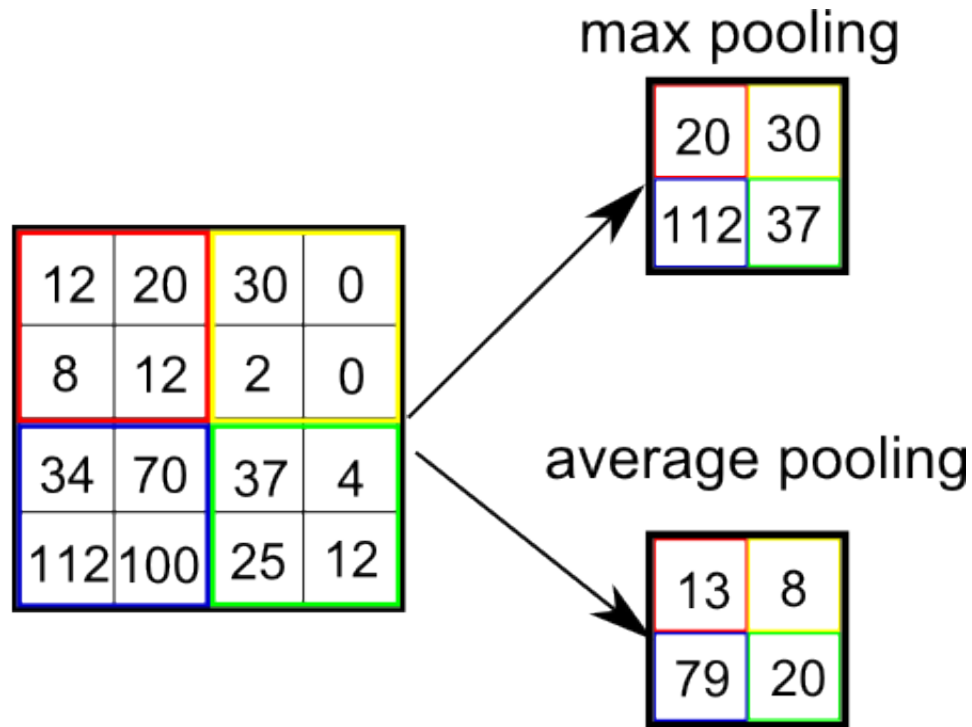
each filter has $5*5*3 + 1 = 76$ params

$\Rightarrow 76*10 = 760$



(+1 for bias)

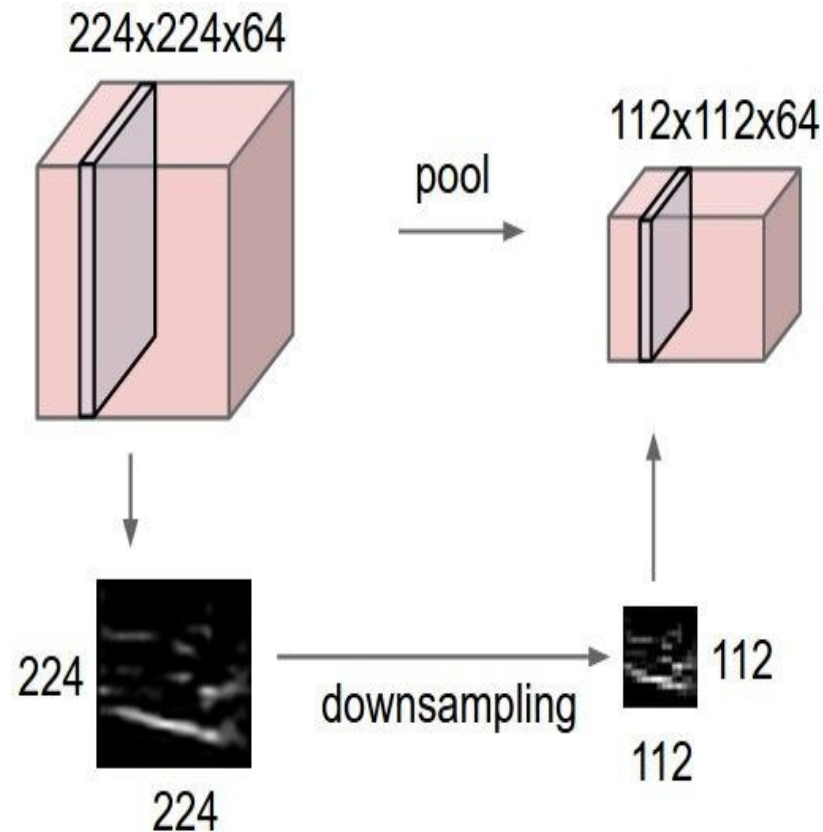
Pooling



Effect = invariance to small translations of the input

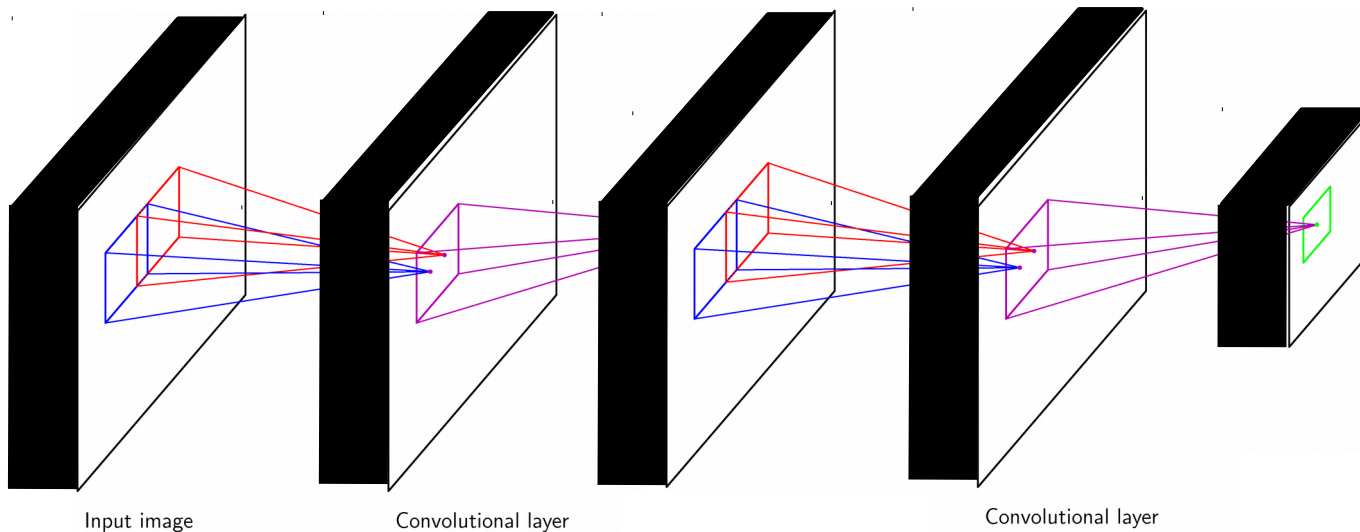
Pooling

- makes the representations smaller and computationally less expensive
- operates over each activation map independently



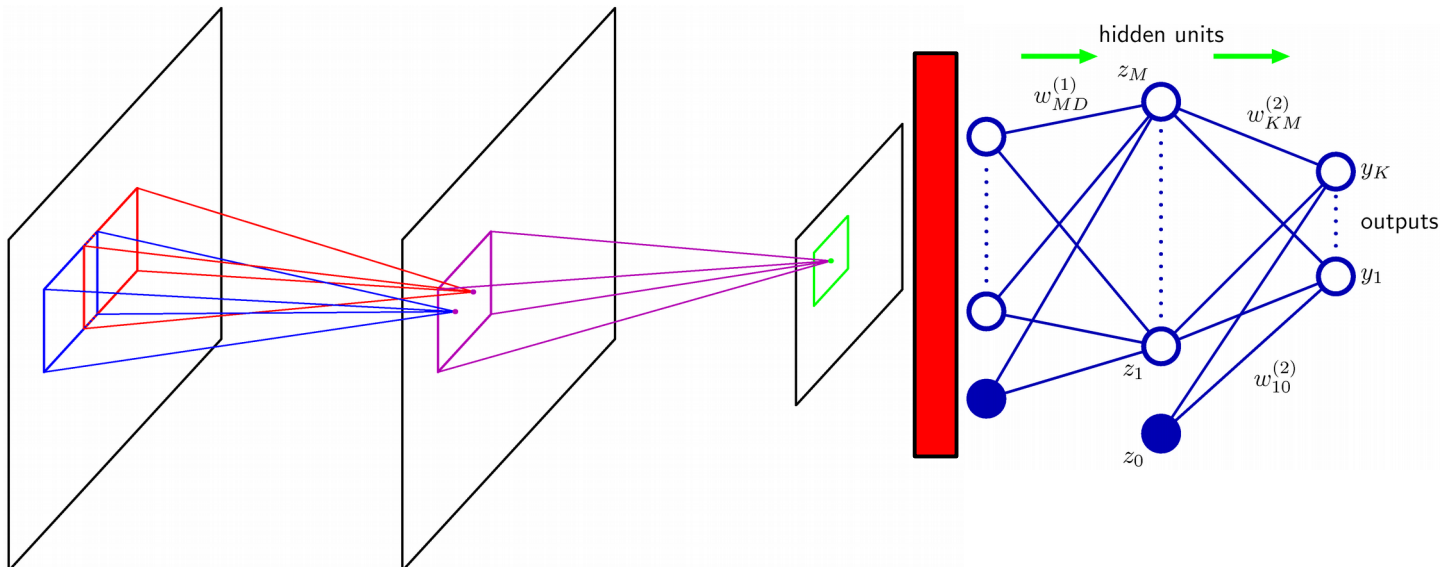
Receptive fields

- “Receptive field” is area in original image impacting a certain unit
 - ▶ Later layers can capture more complex patterns over larger areas
- Receptive field size grows linearly over convolutional layers
 - ▶ If we use a convolutional filter of size $w \times w$, then each layer the receptive field increases by $(w-1)$
- Receptive field size increases exponentially over pooling layers
 - ▶ It is the stride that makes the difference, not pooling vs convolution



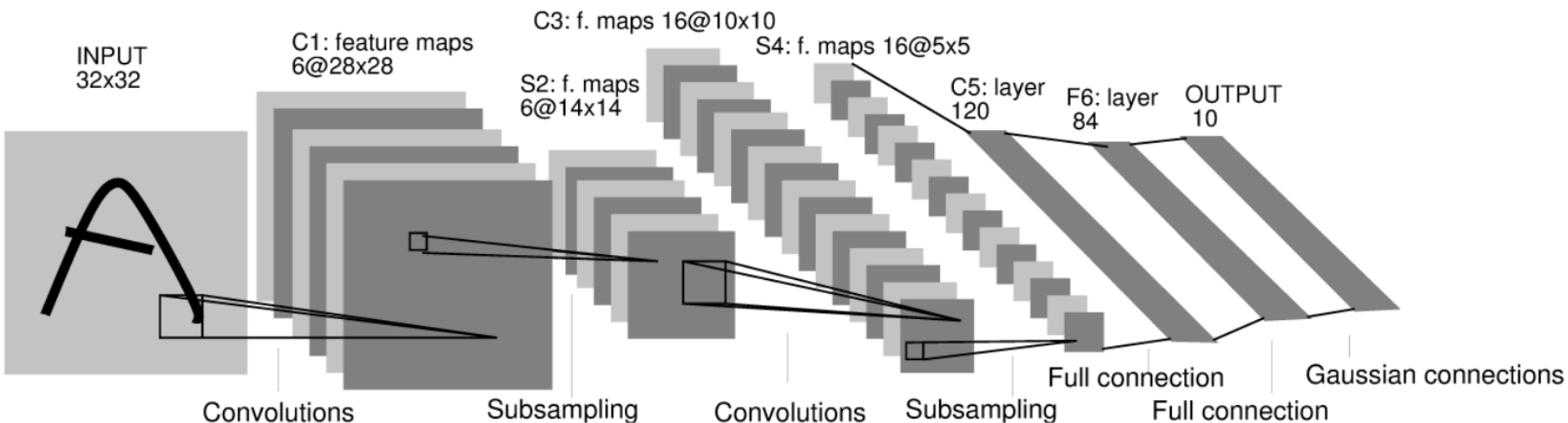
Fully connected layers

- Convolutional and pooling layers typically followed by several “fully connected” (FC) layers, i.e. standard multi-layer network
 - ▶ FC layer connects all units in previous layer to all units in next layer
 - ▶ Assembles all local information into global vectorial representation
- FC layers followed by softmax over outputs to generate distribution over image class labels
- First FC layer that connects response map to vector has many parameters
 - ▶ Conv layer of size $16 \times 16 \times 256$ with following FC layer with 4096 units leads to a connection with 256 million parameters !



Convolutional neural network architectures

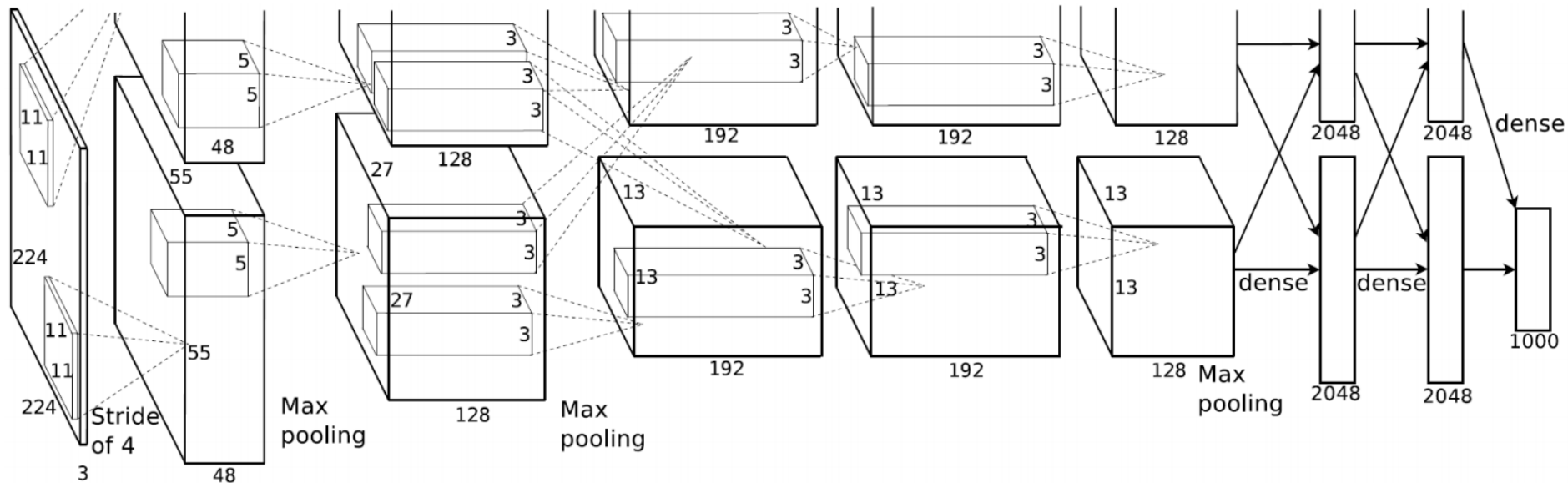
- Surprisingly little difference between today's architectures and those of late eighties and nineties
 - ▶ Convolutional layers, same
 - ▶ Nonlinearities: ReLU dominant now, tanh before
 - ▶ Subsampling: more strided convolution now than max/average pooling



Handwritten digit recognition network. LeCun, Bottou, Bengio, Haffner, Proceedings IEEE, 1998

Convolutional neural network architectures

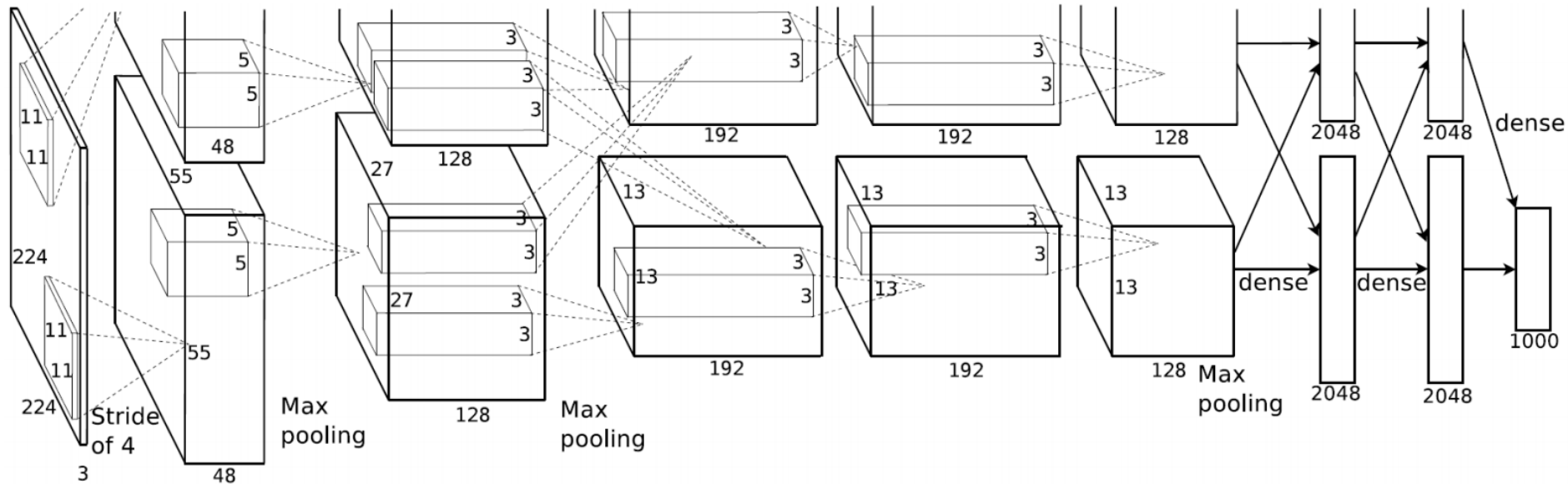
- Recent success with deeper networks
 - ▶ 19 layers in Simonyan & Zisserman, ICLR 2015
 - ▶ Hundreds of layers in residual networks, He et al. ECCV 2016
- More filters per layer: hundreds to thousands instead of tens
- More parameters: tens or hundreds of millions



Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

Other factors that matter

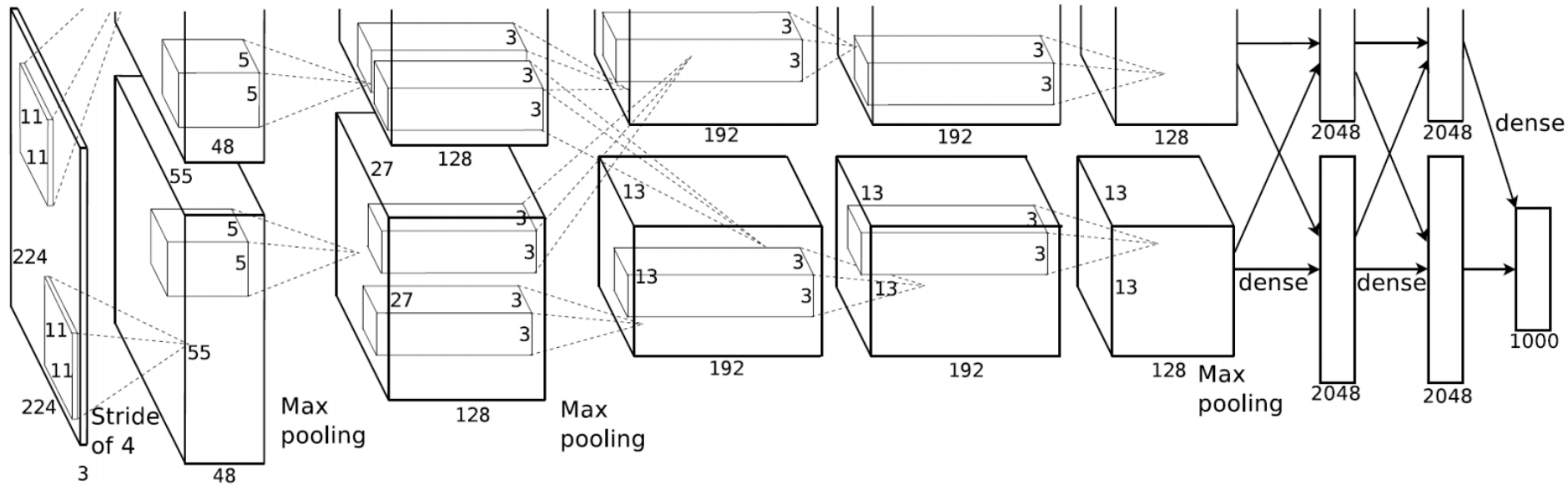
- More training data
 - ▶ 1.2 millions of 1000 classes in ImageNet challenge
 - ▶ 200 million faces in Schroff et al, CVPR 2015
- GPU-based implementations
 - ▶ Massively parallel computation of convolutions
 - ▶ Krizhevsky & Hinton, 2012: six days of training on two GPUs
 - ▶ Rapid progress in GPU compute performance



Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

Understanding convolutional neural network activations

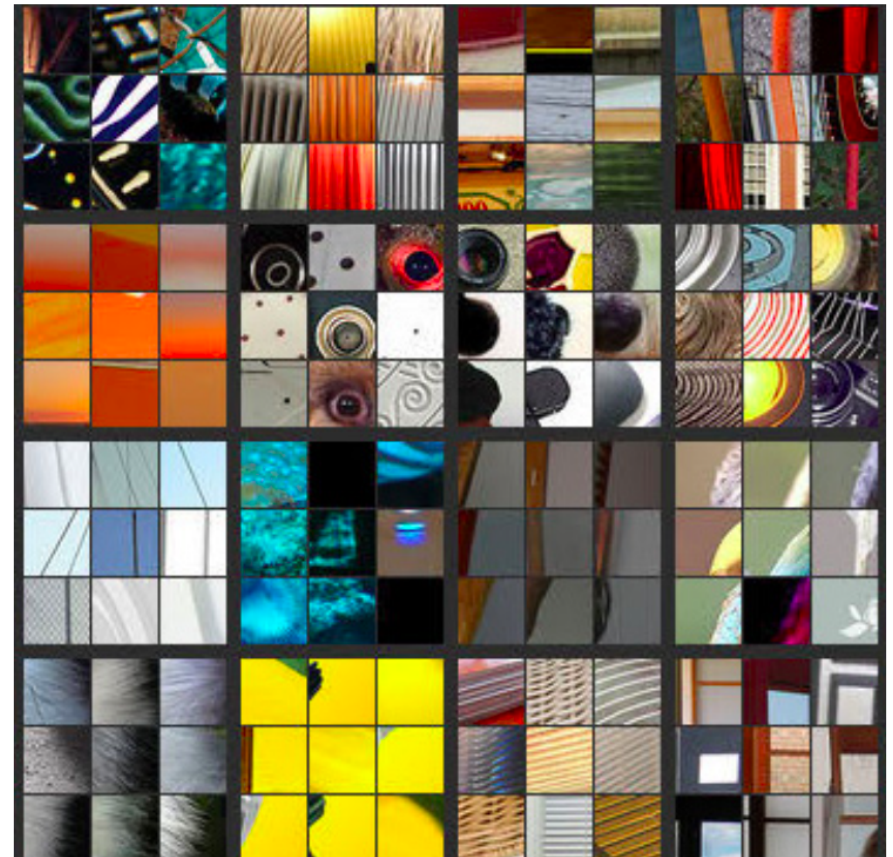
- Architecture consists of
 - ▶ 5 convolutional layers
 - ▶ 2 fully connected layers
- Visualization of patches that yield maximum response for certain units
 - ▶ We will look at each of the 5 convolutional layers



Krizhevsky & Hinton, NIPS 2012, Winning model ImageNet 2012 challenge

Understanding convolutional neural network activations

- Patches generating highest response for a selection of convolutional filters,
 - ▶ Showing 9 patches per filter
 - ▶ Zeiler and Fergus, ECCV 2014
- Layer 1: simple edges and color detectors



- Layer 2: corners, center-surround, ...

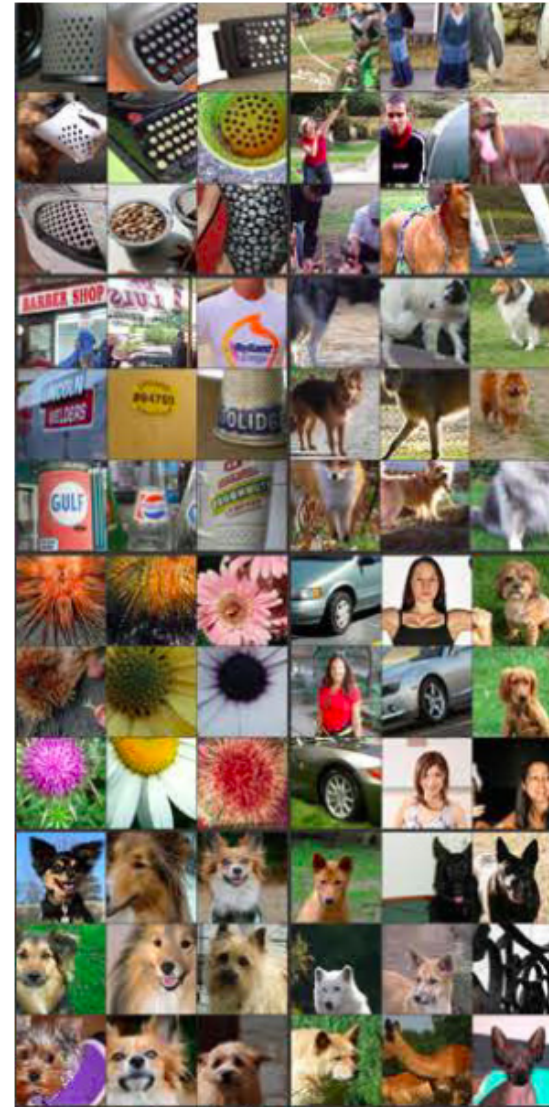
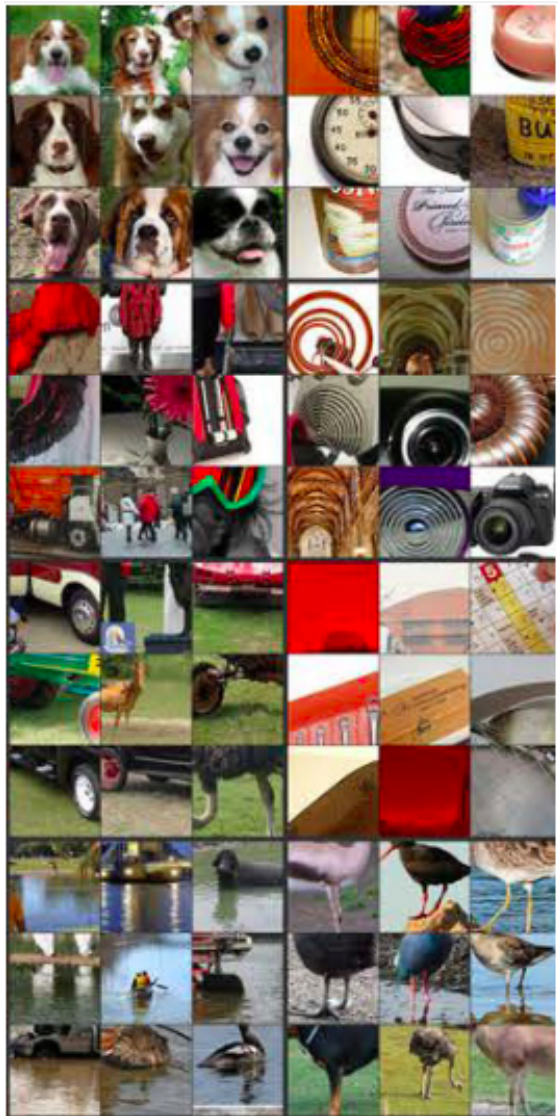
Understanding convolutional neural network activations

- Layer 3: various object parts



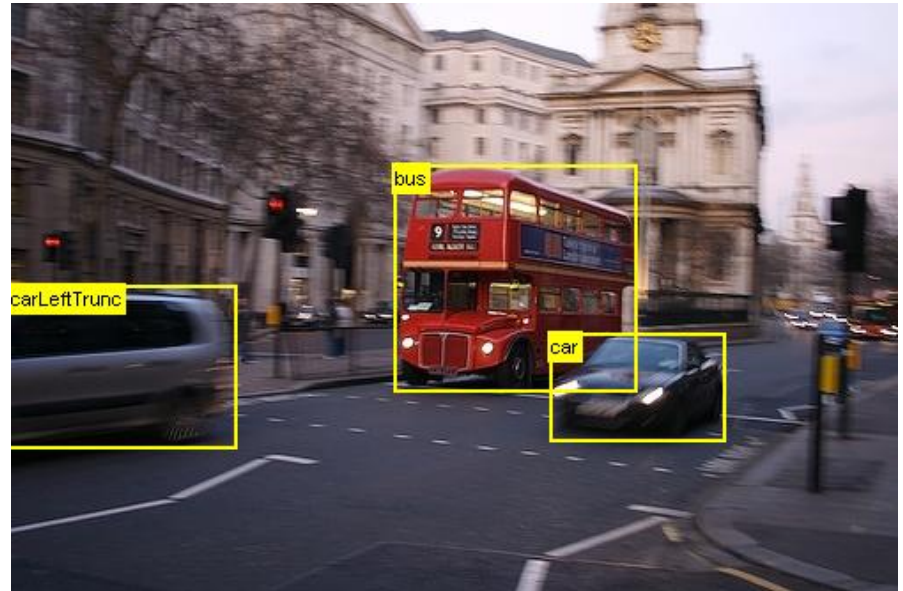
Understanding convolutional neural network activations

- Layer 4+5: selective units for entire objects or large parts of them



Convolutional neural networks for other tasks

- Object category localization

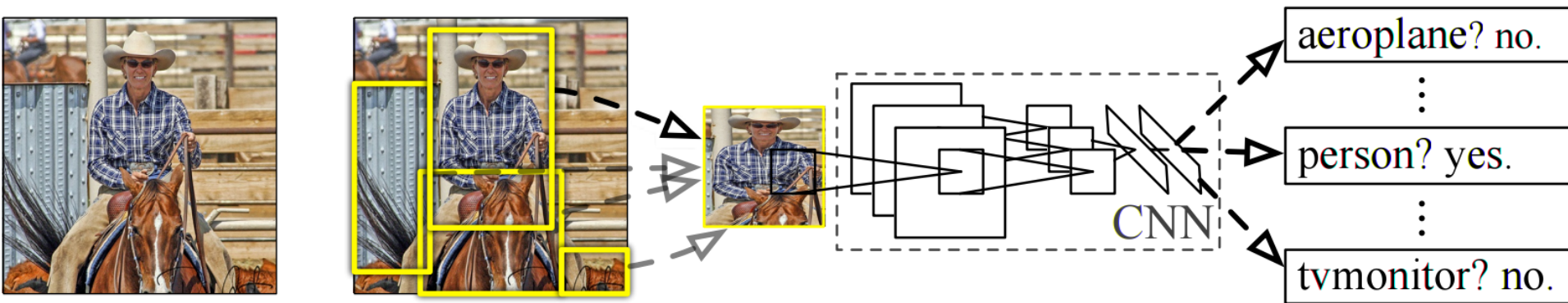


- Semantic segmentation



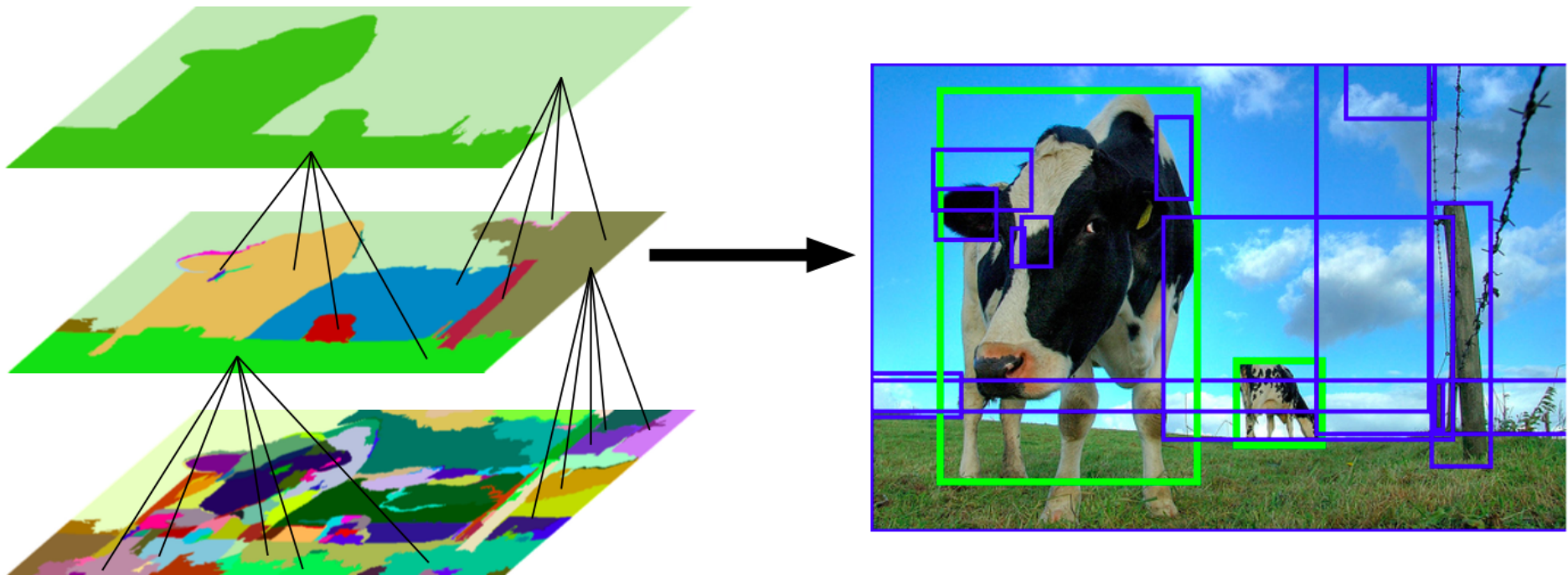
CNNs for object category localization

- Apply CNN image classification model to image sub-windows
 - ▶ For each window decide if it represents a car, sheep, ...
- Resize detection windows to fit CNN input size
- Unreasonably many image regions to consider if applied in naive manner
 - ▶ Use detection proposals based on low-level image contours



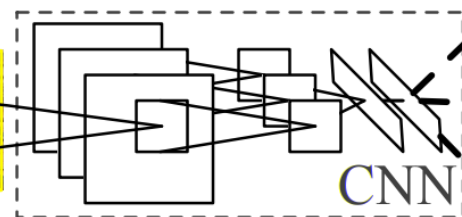
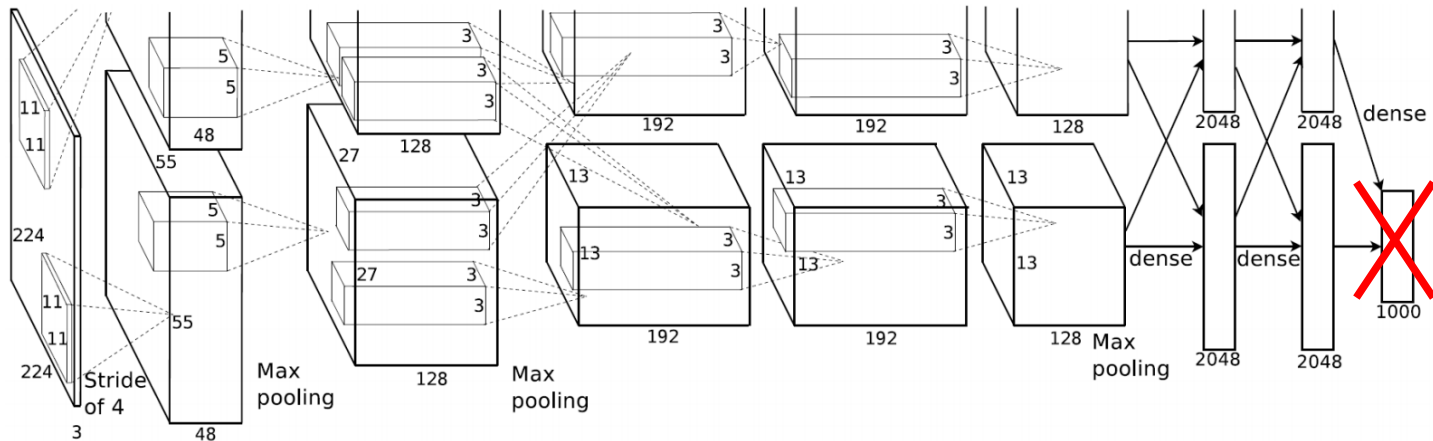
Detection proposal methods

- Many methods exist, some based on learning others not
- Selective search method [Uijlings et al., IJCV, 2013]
 - ▶ Unsupervised multi-resolution hierarchical segmentation
 - ▶ Detections proposals generated as bounding box of segments
 - ▶ 1500 windows per image suffice to cover over 95% of true objects with sufficient accuracy



CNNs for object category localization

- On some datasets too little training data to learn CNN from scratch
 - ▶ Only few hundred objects instances labeled with bounding box
 - ▶ **Pre-train** AlexNet on large ImageNet classification problem
 - ▶ Replace last classification layer with classification over N categories + background
 - ▶ **Fine-tune** CNN weights for classification of detection proposals



- aeroplane? no.
- ⋮
- person? yes.
- ⋮
- tvmonitor? no.

CNNs for object category localization

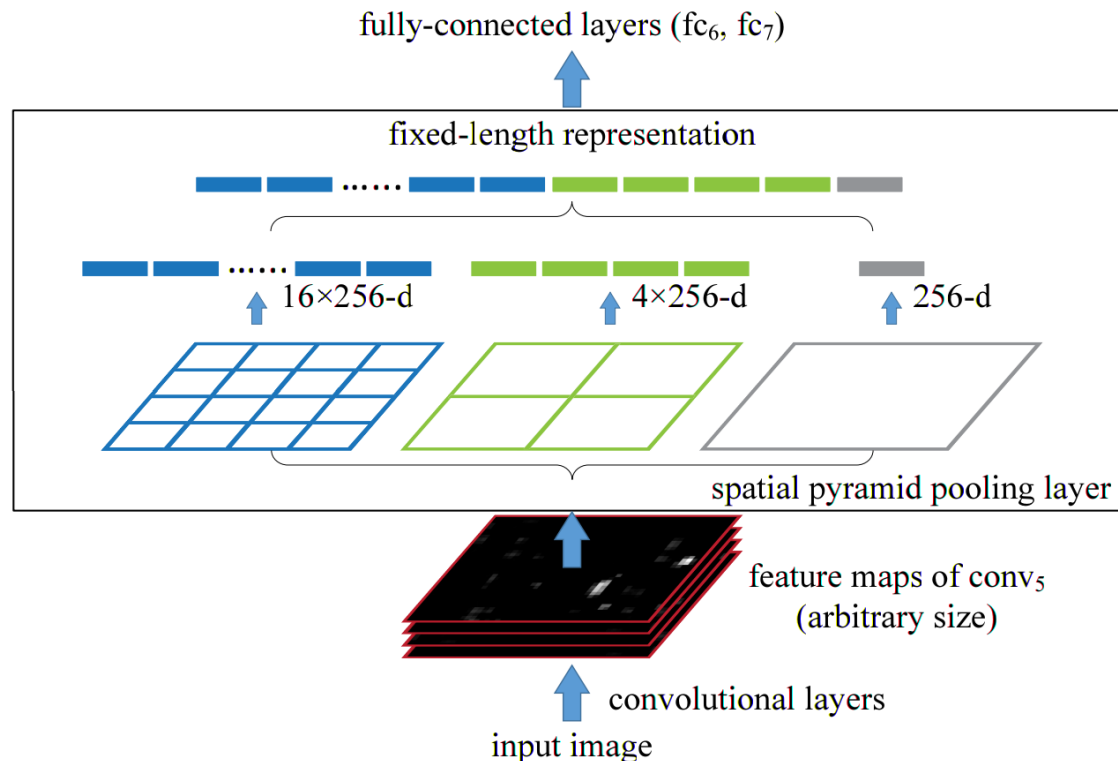
- Comparison with state of the art non-CNN models
 - ▶ Object detection is correct if window has intersection/union with ground-truth window of at least 50%
- Significant increase in performance of 10 points mean-average-precision (mAP)

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20] [†]	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18] [†]	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	71.8	65.8	53.0	36.8	35.9	59.7	60.0	69.9	27.9	50.6	41.4	70.0	62.0	69.0	58.1	29.5	59.4	39.3	61.2	52.4	53.7

Table 1: Detection average precision (%) on VOC 2010 test. R-CNN is most directly comparable to UVA and Regionlets since all methods use selective search region proposals. Bounding-box regression (BB) is described in Section C. At publication time, SegDPM was the top-performer on the PASCAL VOC leaderboard. [†]DPM and SegDPM use context rescoring not used by the other methods.

Efficient object category localization with CNN

- R-CNN recomputes convolutions many times across overlapping regions
- Instead: compute convolutional part only once across entire image
- For each window:
 - ▶ Pool convolutional features using max-pooling into fixed-size representation
 - ▶ Fully connected layers up to classification computed per window



SPP-net, He et al., ECCV 2014

Efficient object category localization with CNN

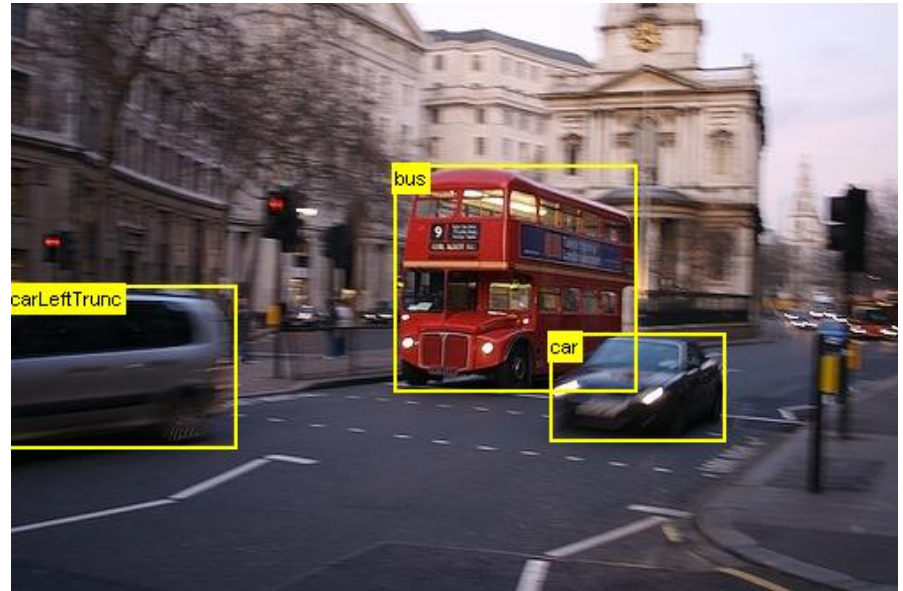
- Refinement: Compute convolutional filters at multiple scales
 - ▶ For given window use scale at which window has roughly size 224x224
- Similar performance as explicit window rescaling, and re-computing convolutional filters
- Speedup of about 2 orders of magnitude

	SPP (1-sc) (ZF-5)	SPP (5-sc) (ZF-5)	R-CNN (ZF-5)
ftfc ₇	54.5	<u>55.2</u>	55.1
ftfc ₇ bb	58.0	59.2	59.2
conv time (GPU)	0.053s	0.293s	14.37s
fc time (GPU)	0.089s	0.089s	0.089s
total time (GPU)	0.142s	0.382s	14.46s
speedup (<i>vs.</i> RCNN)	102 ×	38 ×	-

Table 10: Detection results (mAP) on Pascal VOC 2007, using the same pre-trained model of SPP (ZF-5).

Convolutional neural networks for other tasks

- Object category localization

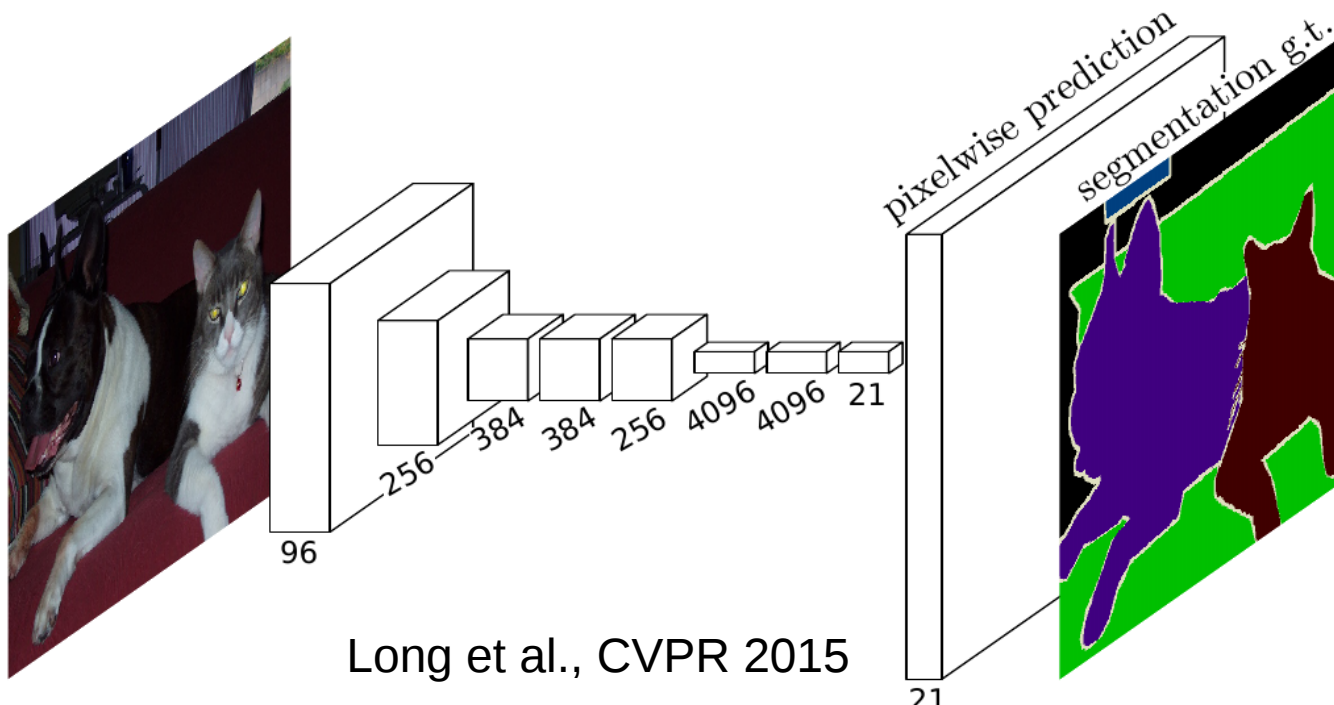


- Semantic segmentation



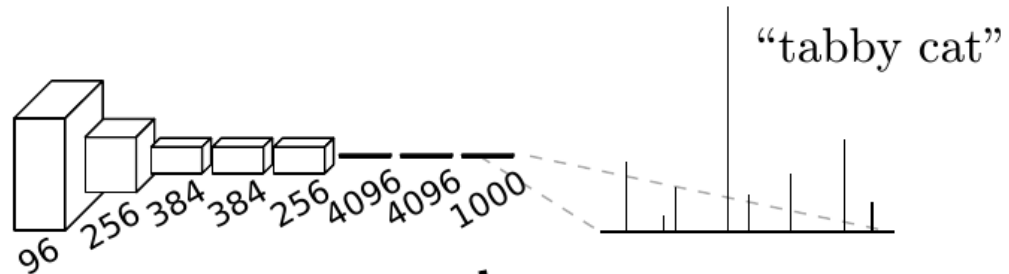
Application to semantic segmentation

- Assign each pixel to an object or background category
 - ▶ Consider running CNN on small image patch to determine its category
 - ▶ Train by optimizing per-pixel classification loss
- Similar to SPP-net: want to avoid wasteful computation of convolutional filters
 - ▶ Compute convolutional layers once per image
 - ▶ Here all local image patches are at the same scale
 - ▶ Many more local regions: dense, at every pixel

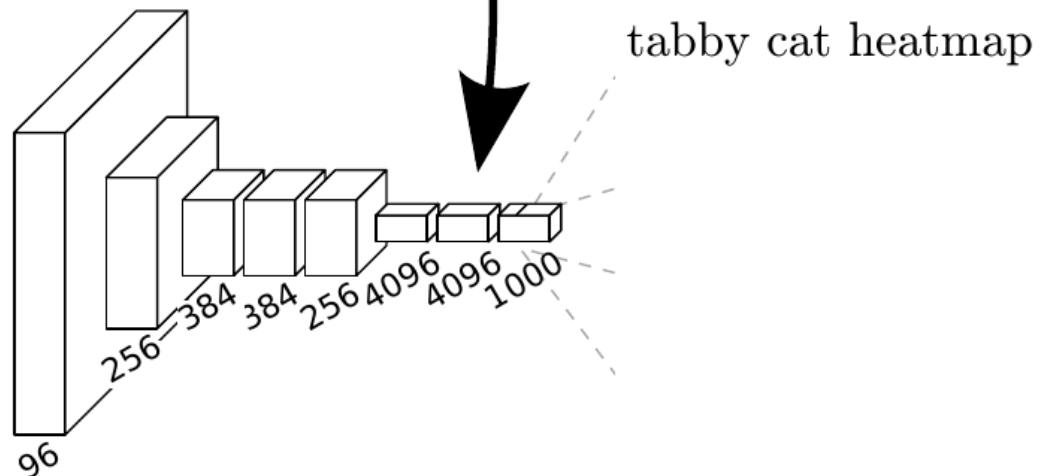


Application to semantic segmentation

- Interpret fully connected layers as 1x1 sized convolutions
 - ▶ Function of features in previous layer, but only at own position
 - ▶ Still same function is applied at all positions
- Five sub-sampling layers reduce the resolution of output map by factor 32

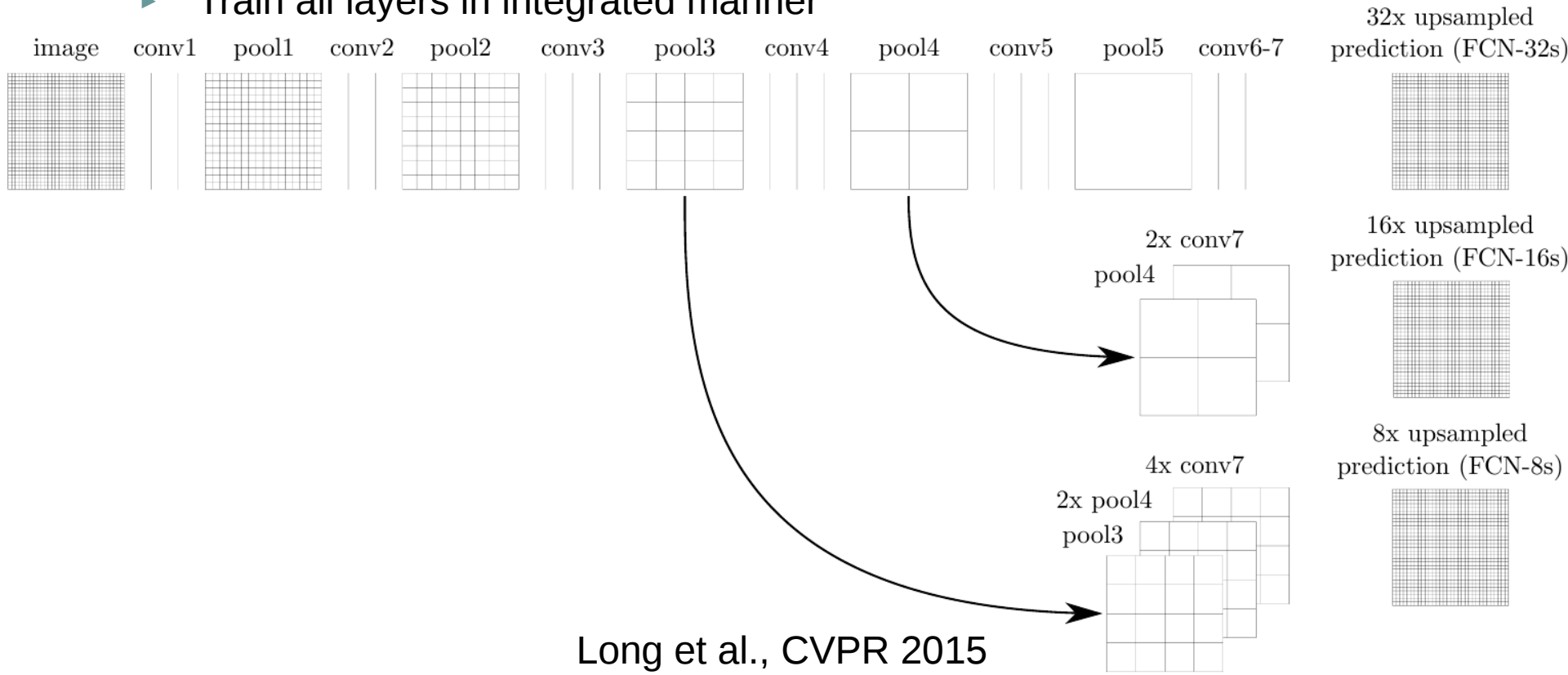


convolutionalization



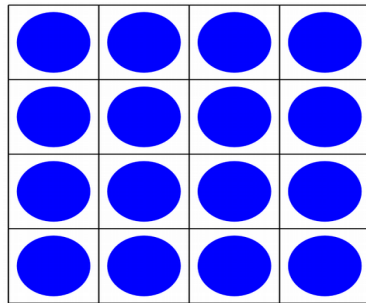
Application to semantic segmentation

- Idea 1: up-sampling via bi-linear interpolation
 - ▶ Gives blurry predictions
- Idea 2: weighted sum of response maps at different resolutions
 - ▶ Upsampling of the later and coarser layer
 - ▶ Concatenate fine layers and upsampled coarser ones for prediction
 - ▶ Train all layers in integrated manner

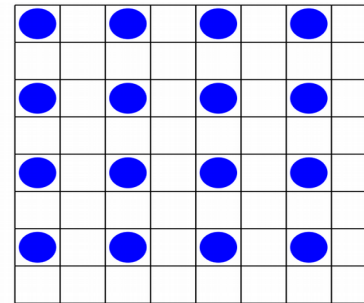


Upsampling of coarse activation maps

- Simplest form: use bilinear interpolation or nearest neighbor interpolation
 - ▶ Note that these can be seen as upsampling by zero-padding, followed by convolution with specific filters, no channel interactions
- Idea can be generalized by learning the convolutional filter
 - ▶ No need to hand-pick the interpolation scheme
 - ▶ Can include channel interactions, if those turn out be useful



Bi-linear: $\frac{1}{4} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$



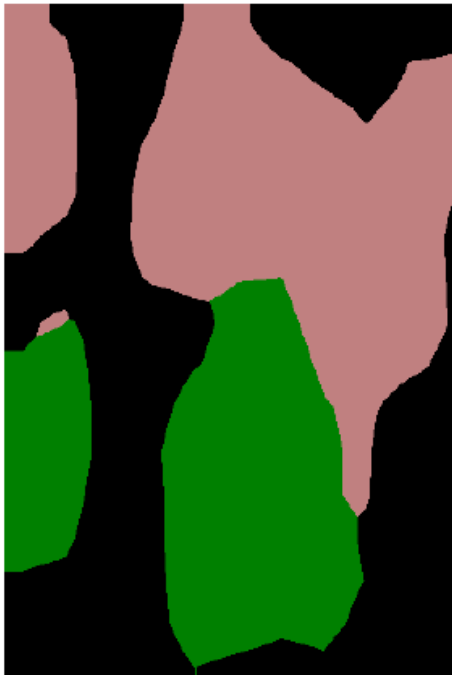
Nearest neighbor: $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

- Resolution-increasing counterpart of strided convolution
 - ▶ Average and max pooling can be written in terms of convolutions
 - ▶ See: “Convolutional Neural Fabrics”, Saxena & Verbeek, NIPS 2016.

Application to semantic segmentation

- Results obtained at different resolutions
 - ▶ Detail better preserved at finer resolutions

FCN-32s



FCN-16s



FCN-8s



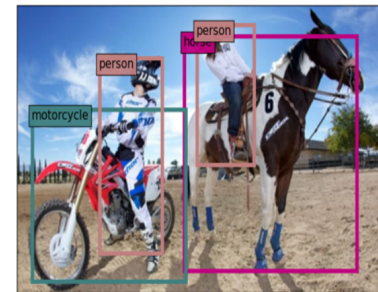
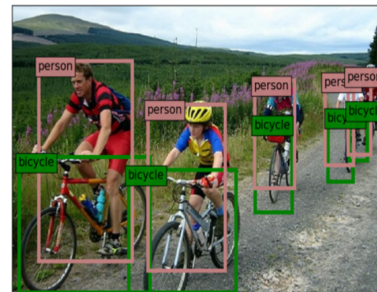
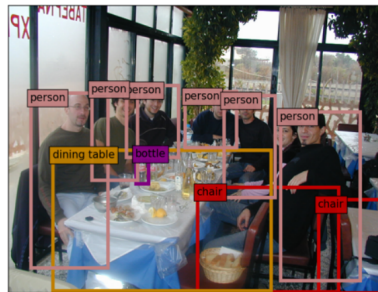
Ground truth



Semantic segmentation: further improvements

- Beyond independent prediction of pixel labels
 - ▶ Integrate conditional random field (CRF) models with CNN

Zheng et al., ICCV'15



- Using more sophisticated upsampling schemes to maintain high-resolution signals

Kokkinos, arXiv 2016

Saxena & Verbeek
NIPS 2016

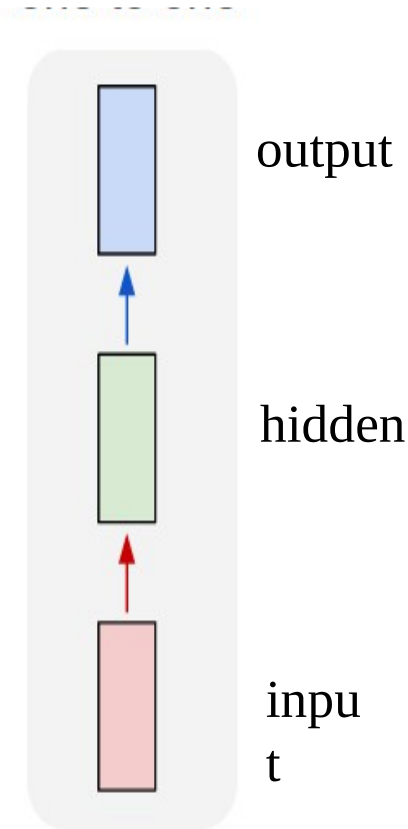
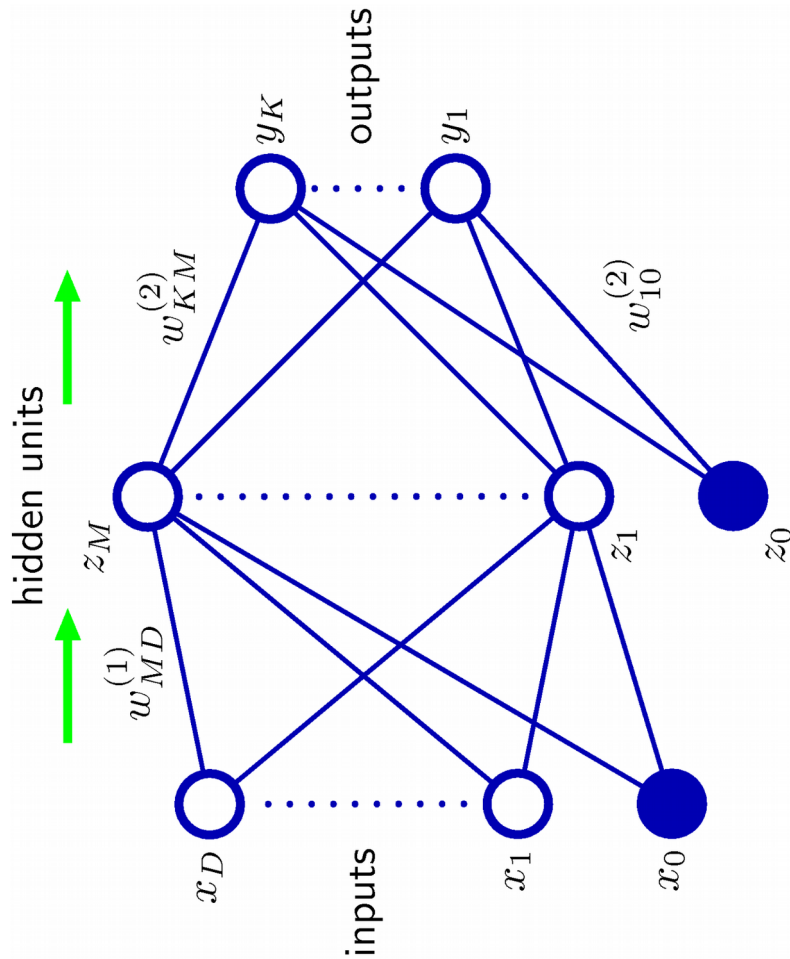


Summary feed-forward neural networks

- Construction of complex functions with circuits of simple building blocks
 - ▶ Linear function of previous layers
 - ▶ Scalar non-linearity
- Learning via back-propagation of error gradient throughout network
 - ▶ Need directed acyclic graph
- Convolutional neural networks (CNNs) extremely useful for image data
 - ▶ State-of-the-art results in a wide variety of computer vision tasks
 - ▶ Spatial invariance of processing (also useful for video, audio, ...)
 - ▶ Stages of aggregation of local features into more complex patterns
 - ▶ Same weights shared for many units organized in response maps

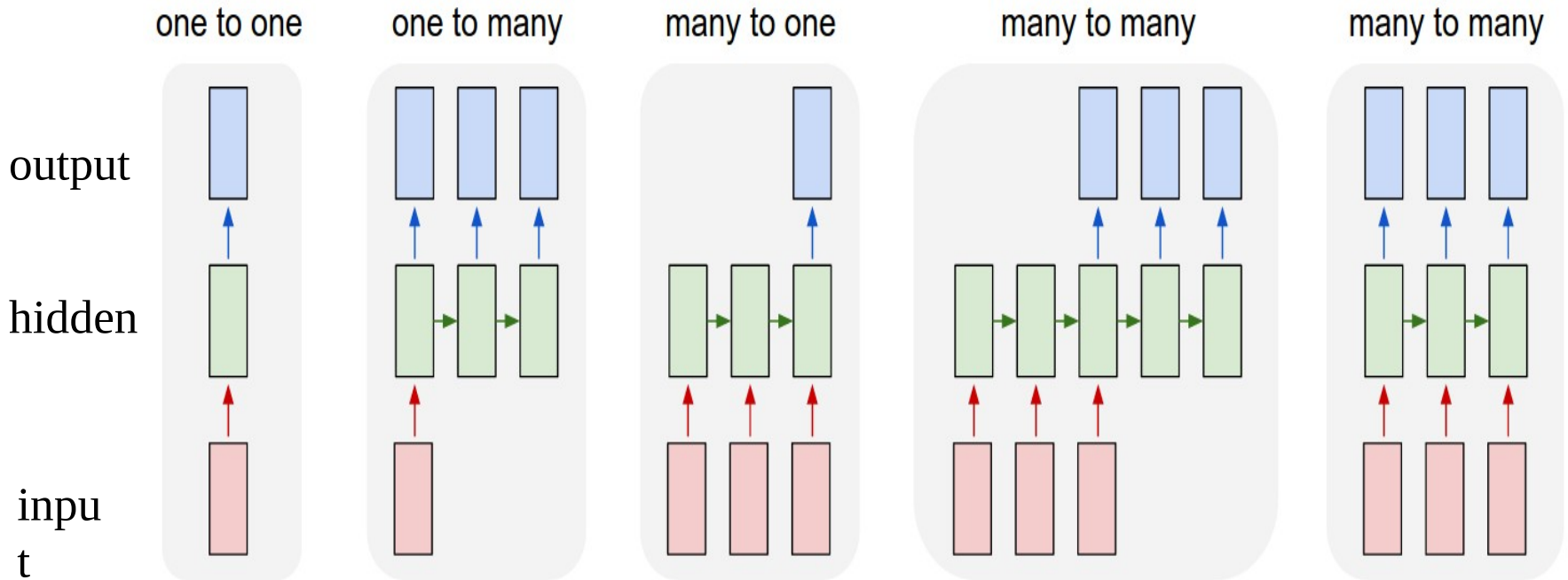
Modeling sequential data

- Compact schematic drawing of standard multi-layer perceptron (MLP)



Modeling sequential data

- So far we mostly considered prediction tasks with a fixed sized input/output
 - ▶ Classification: one image, one class label
 - ▶ Segmentation: one image, per pixel labels
- Many prediction problems have a sequential nature to them
 - ▶ Either in input, in output, or both
 - ▶ Both may vary in length from one example to another



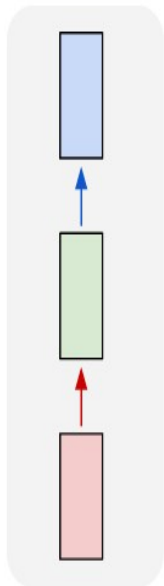
Modeling sequential data

- Image captioning
 - ▶ Input: an image
 - ▶ Output: natural language description

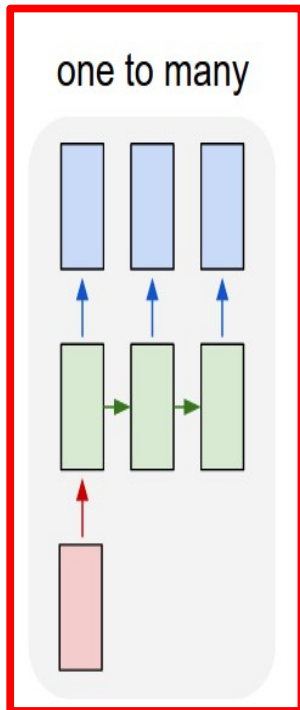


a brown dog is running through the grass

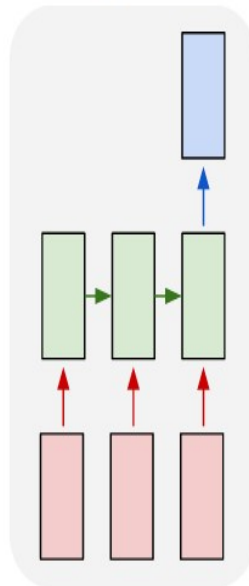
one to one



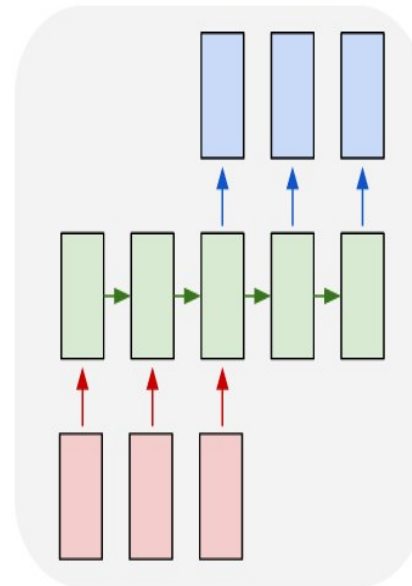
one to many



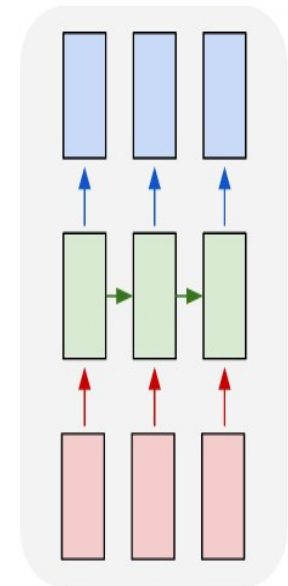
many to one



many to many



many to many



Modeling sequential data

- Natural language processing
 - ▶ Input: a sentence
 - ▶ Output: user rating



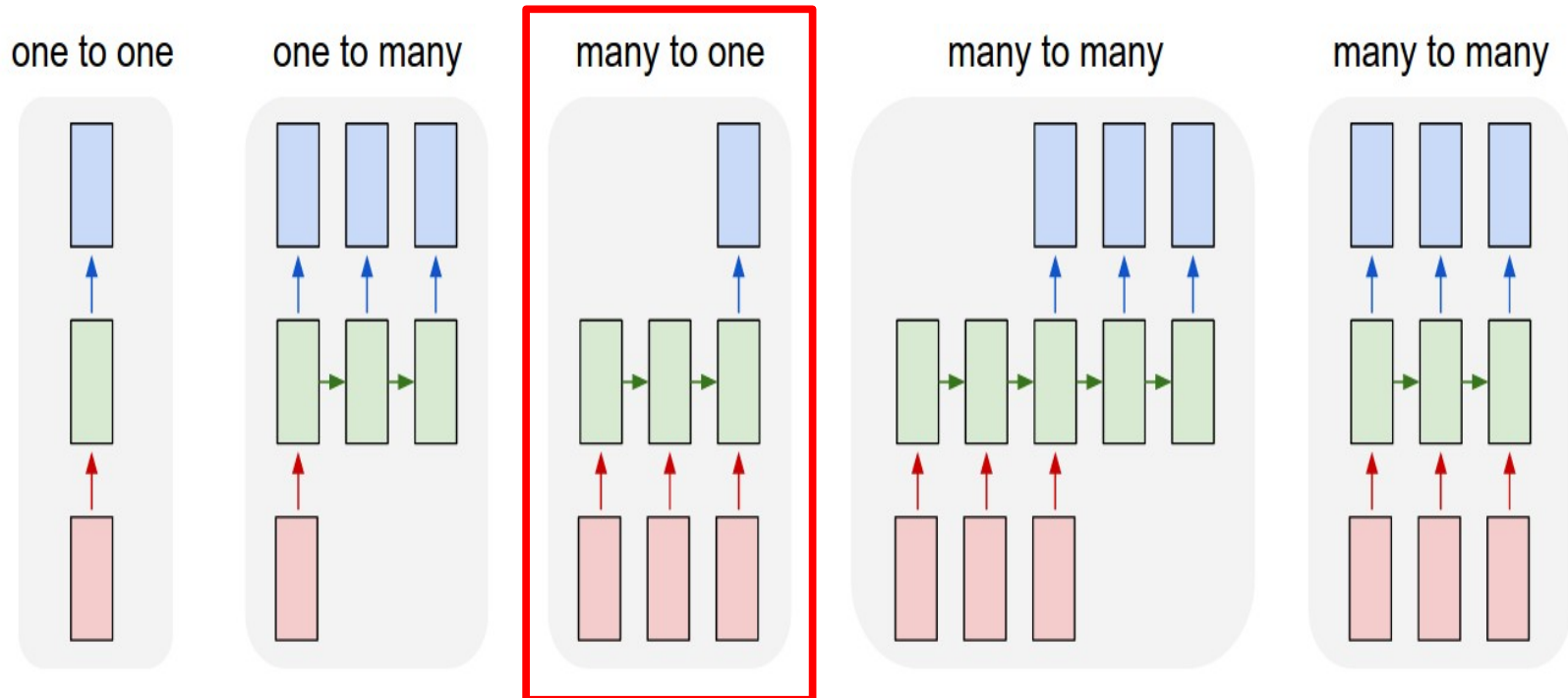
I liked it...



Author: [JustMeOnline](#) from Portugal

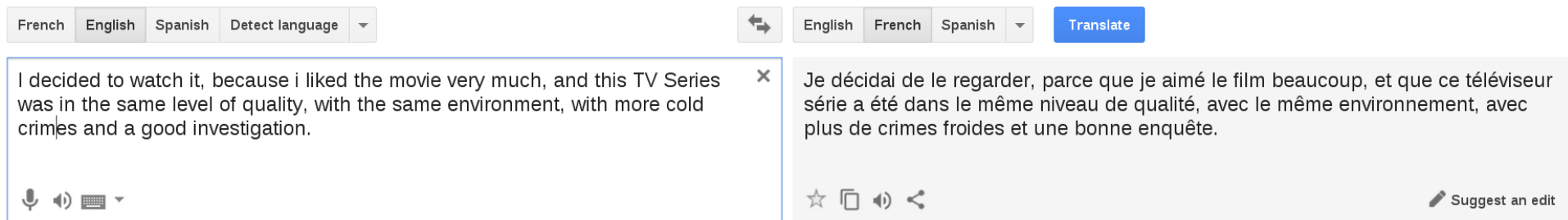
14 November 2014

I decided to watch it, because i liked the movie very much, and this TV Series was in the same level of quality, with the same environment, with more cold crimes and a good investigation...



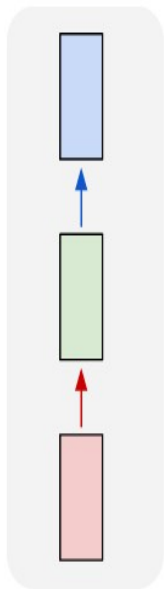
Modeling sequential data

- Machine translation of text from one language to another
 - ▶ Sequences of different length on input and output
- Encoder-decoder: source encoded to latent state, then decode to target

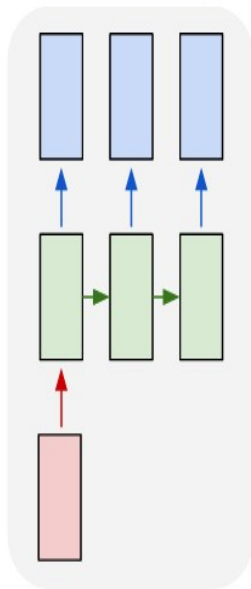


The screenshot shows a web-based machine translation interface. On the left, there are language selection buttons for French, English, and Spanish, and a 'Detect language' dropdown. On the right, there are buttons for English, French, and Spanish, and a 'Translate' button. The input text on the left is: "I decided to watch it, because i liked the movie very much, and this TV Series was in the same level of quality, with the same environment, with more cold crimes and a good investigation." The output text on the right is: "Je décidai de le regarder, parce que je aimé le film beaucoup, et que ce téléviseur série a été dans le même niveau de qualité, avec le même environnement, avec plus de crimes froides et une bonne enquête." Below the input and output text are icons for voice input, volume, keyboard, and a 'Suggest an edit' link.

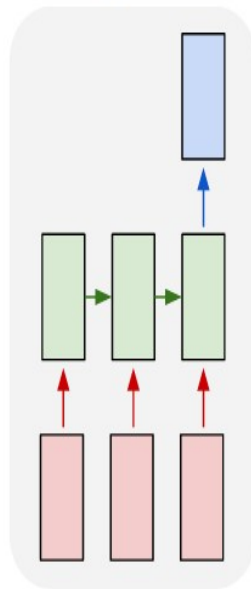
one to one



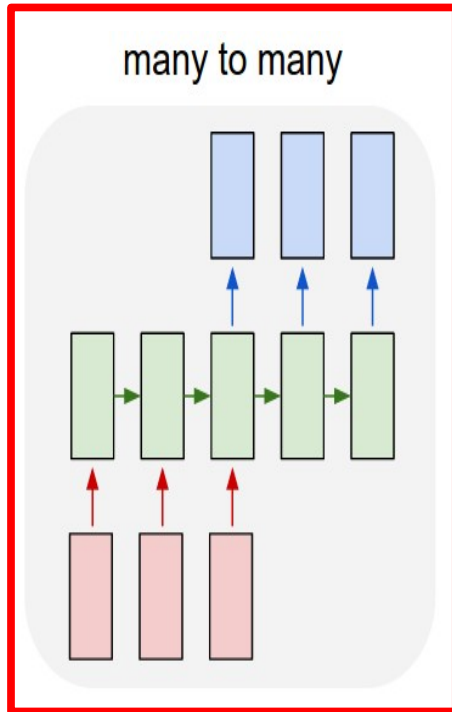
one to many



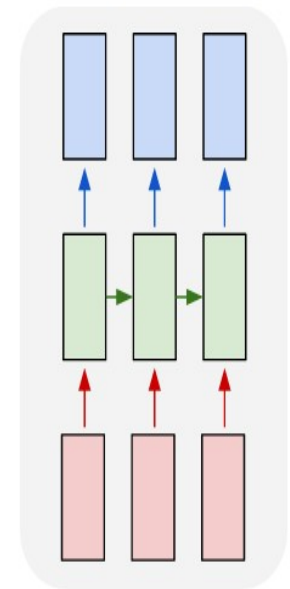
many to one



many to many

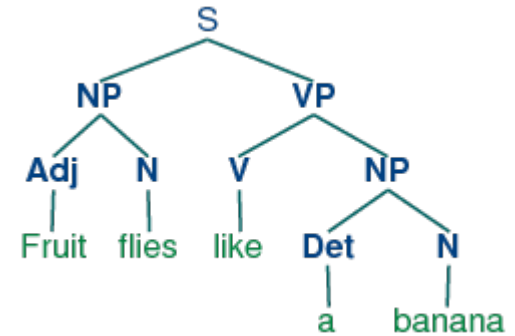


many to many

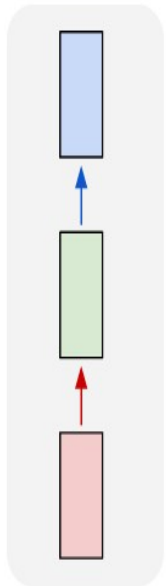


Modeling sequential data

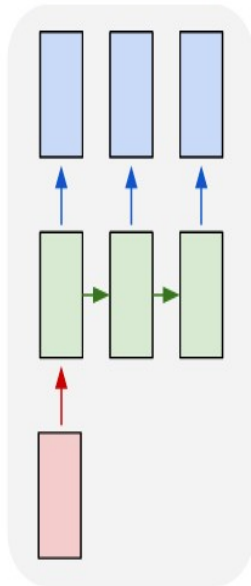
- Part of speech tagging
 - ▶ Input: a sentence
 - ▶ Output: a part of speech tag for each word
- Input and output sequence of same length



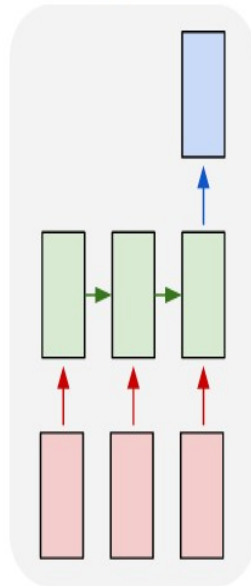
one to one



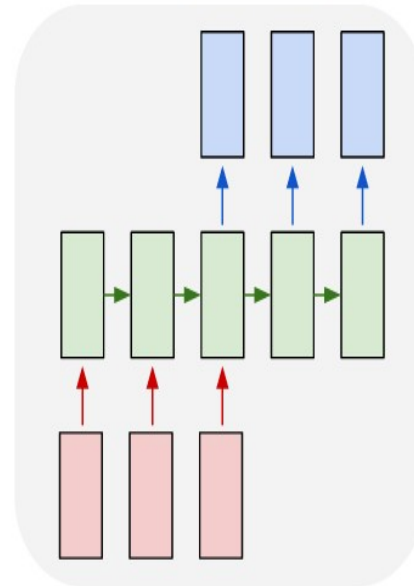
one to many



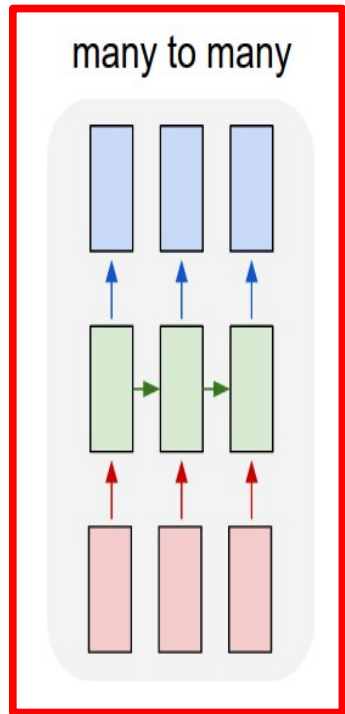
many to one



many to many



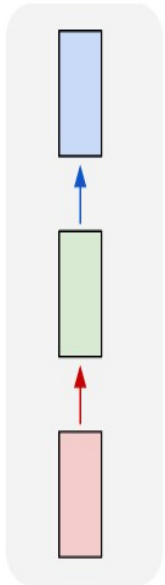
many to many



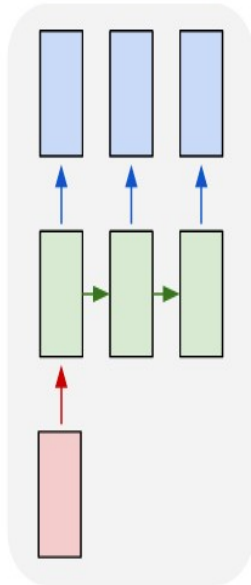
Modeling sequential data

- Using k-order Markov chains over input and/or output sequences limits memory to only k time steps in the past
- A layer of hidden units allows to build up a representation of the sequence as a whole over time, even using a first-order system
 - ▶ Inputs and/or outputs influence hidden state update, carried on to future time steps

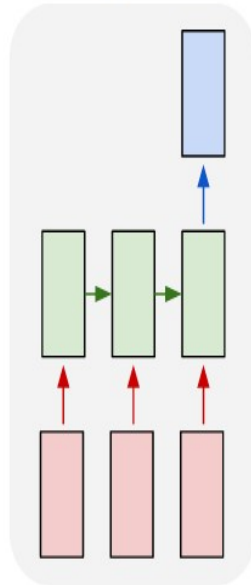
one to one



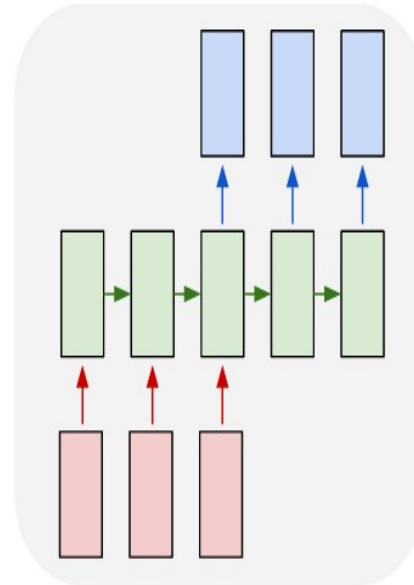
one to many



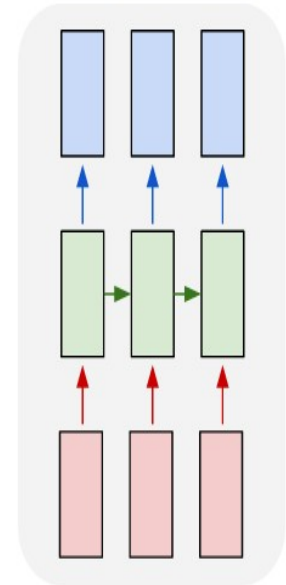
many to one



many to many



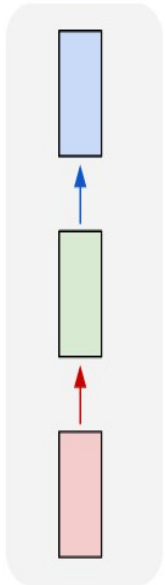
many to many



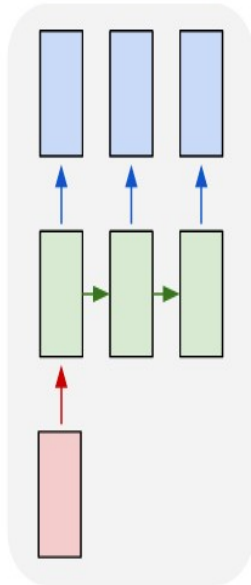
Recurrent neural networks

- Recurrent computation of hidden units from one time step to the next
 - ▶ Time-invariant recurrence makes it applicable to arbitrarily long sequences
- Similar ideas of parameter sharing used in
 - ▶ (Hidden) Markov models or Kalman filters for arbitrarily long sequences
 - ▶ Across space in convolutional neural networks

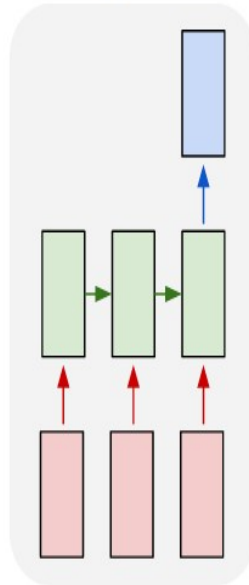
one to one



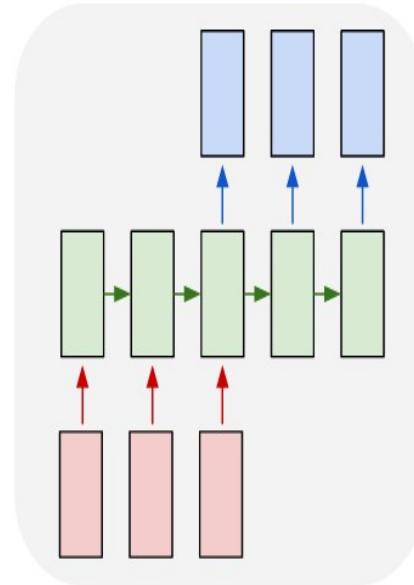
one to many



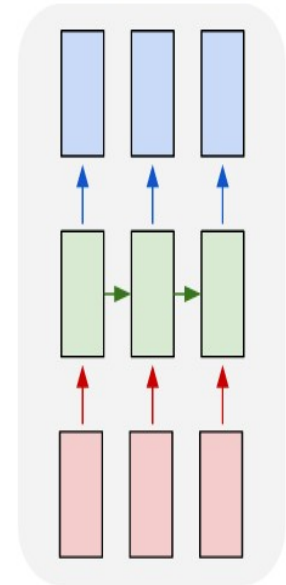
many to one



many to many



many to many



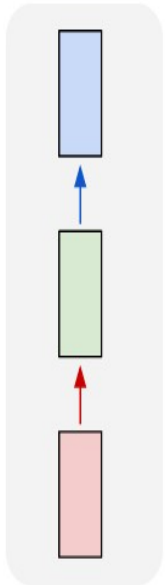
Recurrent neural networks

- Basic example for many-to-many prediction
 - ▶ Hidden state linear function of current input and previous hidden state, followed by point-wise non-linearity
 - ▶ Output is linear function of current hidden state, followed by point-wise non-linearity

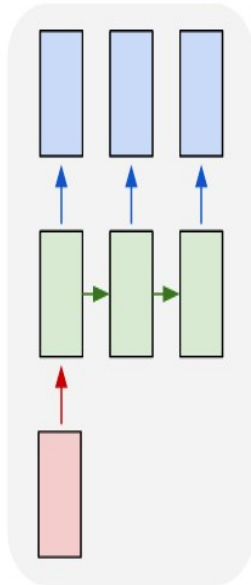
$$z_t = \phi(A x_t + B z_{t-1})$$

$$y_t = \psi(C z_t)$$

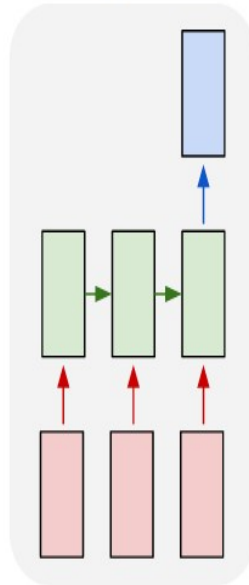
one to one



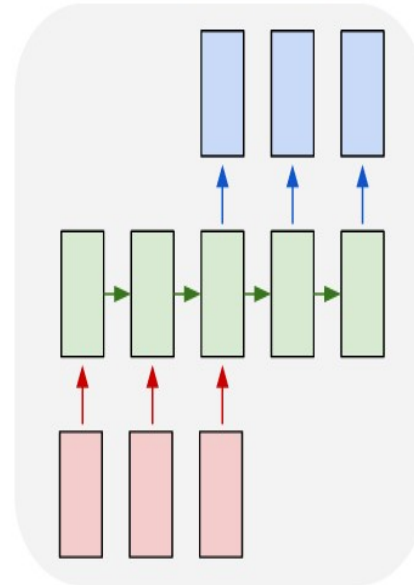
one to many



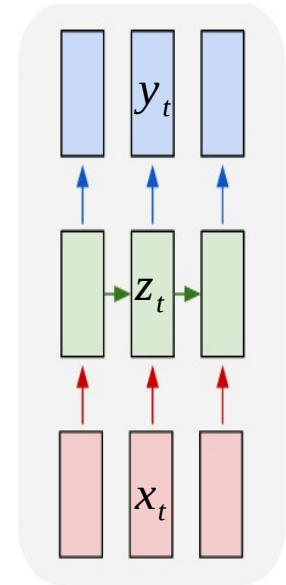
many to one



many to many



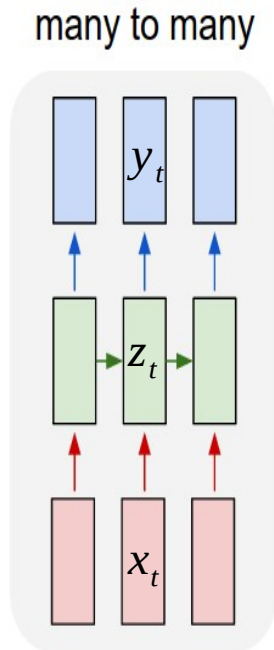
many to many



Recurrent neural network diagrams

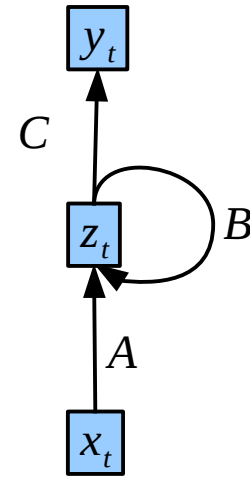
- Two graphical representations are used

“Unfolded” flow diagram



$$z_t = \phi(Ax_t + Bz_{t-1})$$
$$y_t = \psi(Cz_t)$$

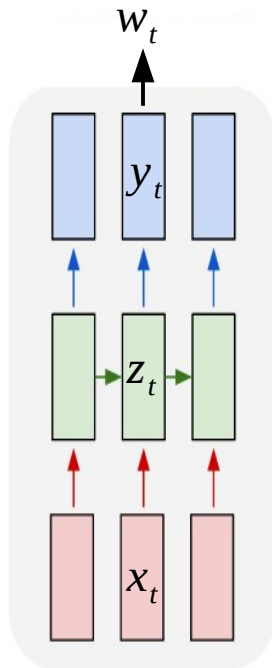
Recurrent flow diagram



- Unfolded representation shows that we still have an acyclic directed graph
 - ▶ Size of the graph (horizontally) is variable, given by sequence length
 - ▶ Weights are shared across horizontal replications
- Gradient computation via back-propagation still works
 - ▶ Referred to as “back-propagation through time” (Pearlmutter, 1989)

Recurrent neural network diagrams

- Deterministic feed-forward network from inputs to outputs
- Predictive model over output sequence is obtained by defining a distribution over outputs given y
 - ▶ For example: probability of a word given via softmax of word score



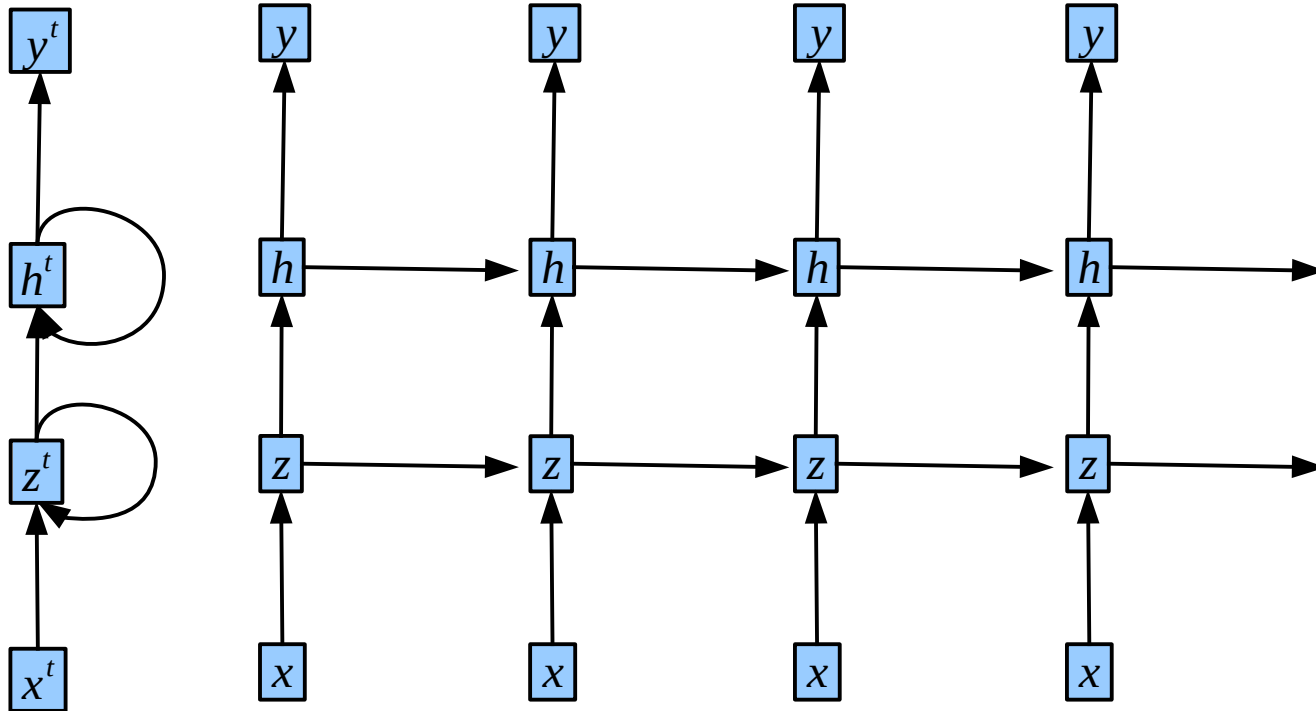
$$z_t = \phi(A x_t + B z_{t-1})$$

$$y_t = \psi(C z_t)$$

$$p(w_t = k) = \frac{\exp y_{tk}}{\sum_{v=1}^V \exp y_{tv}}$$

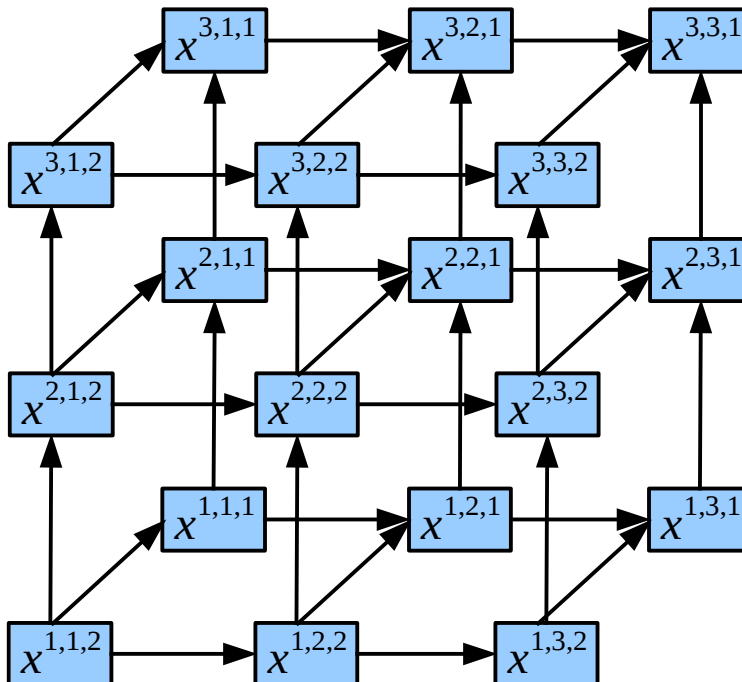
More topologies: “deep” recurrent networks

- Instead of a recurrence across a single hidden layer, consider a recurrence across a multi-layer architecture
- Yet another feed-forward network, with weight sharing over time



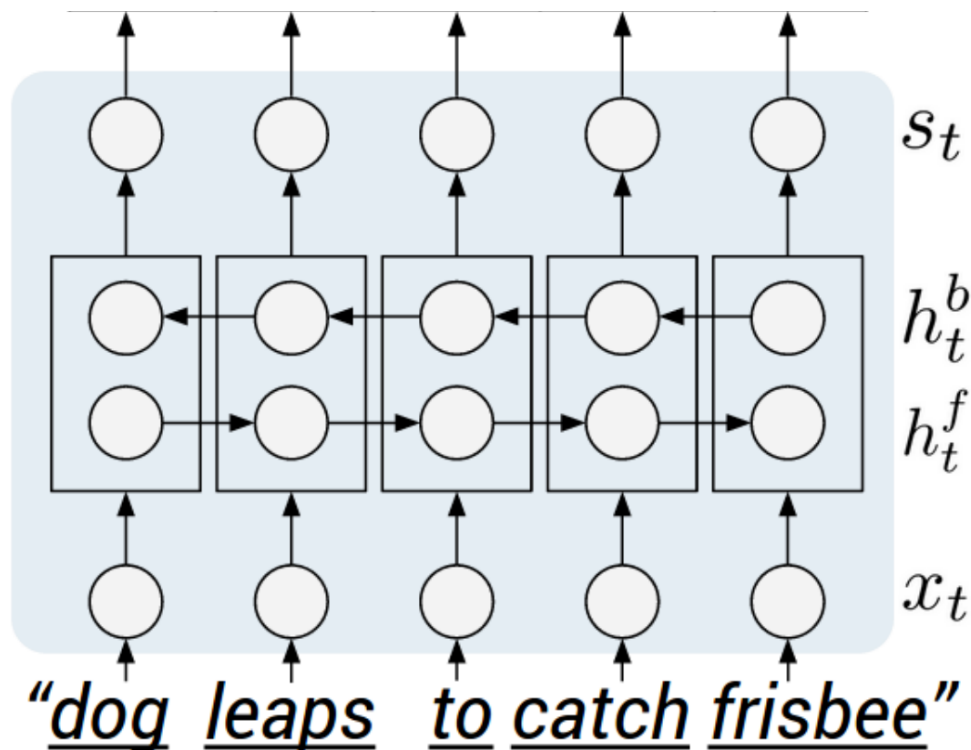
More topologies: multi-dimensional recurrent networks

- Instead of a recurrence across a single (time) axis, consider a recurrence across a multi-dimensional grid
- For example: axis aligned directed edges
 - ▶ Each node receives input from predecessors, one for each dimension



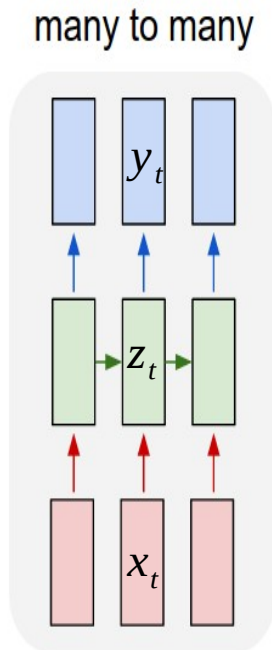
More topologies: bidirectional recurrent neural networks

- Use left and right contextual representation at each time step
 - ▶ Two recurrences, one in each direction
 - ▶ Aggregate output from both directions at each time step for output
- Not Possible on output sequence since it needs to be predicted/generated, and is thus not available for the backwards recurrence



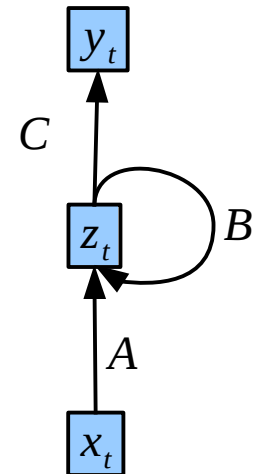
More topologies: output feedback loops

- So far the element in the output sequence at time t was independently drawn given the state at time t
 - ▶ State at time t depends on the entire input sequence up to time t
 - ▶ No dependence on the output sequence produced so far
- Problematic when there are strong regularities in output, eg character or words sequences in natural language



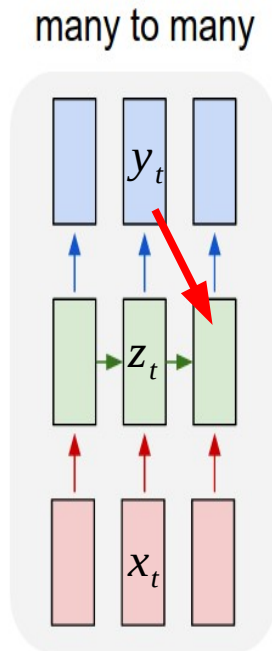
$$z_t = \phi(Ax_t + Bz_{t-1})$$

$$y_t = \psi(Cz_t)$$



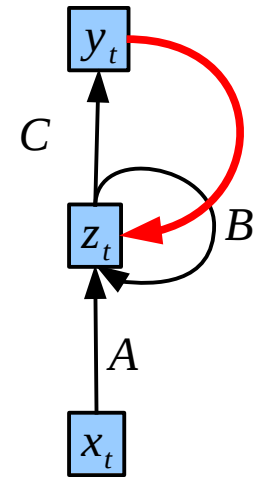
More topologies: output feedback loops

- To introduce dependence on output sequence, we add a **feedback loop** from the output to the hidden state



$$z_t = \phi(Ax_t + Bz_{t-1})$$

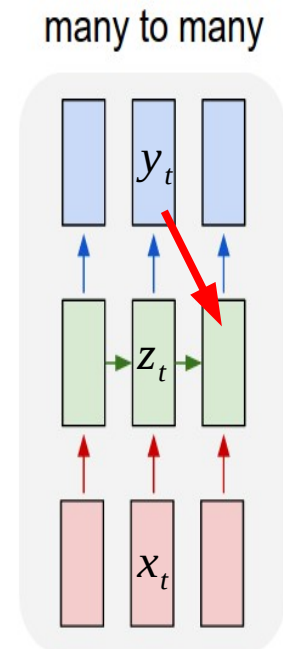
$$y_t = \psi(Cz_t)$$



- Without output-feedback, the state evolution is a deterministic non-linear dynamical system
- With output feedback, the state evolution becomes a **stochastic non-linear dynamical system**
 - ▶ Caused by the stochastic output, which flows back into the state update

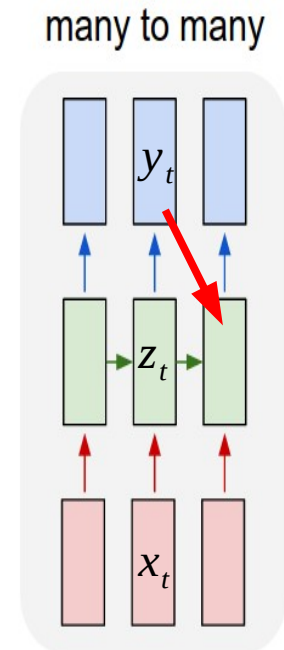
How do we generate data from an RNN ?

- RNN gives a distribution over output sequences
 - ▶ How can we sample from this distribution ?
 - ▶ What else can we do with it ?
- Sequential sampling of output sequence, for each time-step t
 - ▶ Compute state from current input and previous state and output
 - ▶ Compute distribution on current output symbol
 - ▶ Sample output symbol
- Likelihood of sequence given by product of symbol likelihoods
- Compute maximum likelihood sequence?
 - ▶ Not feasible since output symbol impacts state
- Marginal distribution on n -th symbol
 - ▶ Not feasible: marginalize over exponential nr sequences
- Marginal probability of a symbol appearing anywhere in seq.
 - ▶ Not feasible: average over all marginals



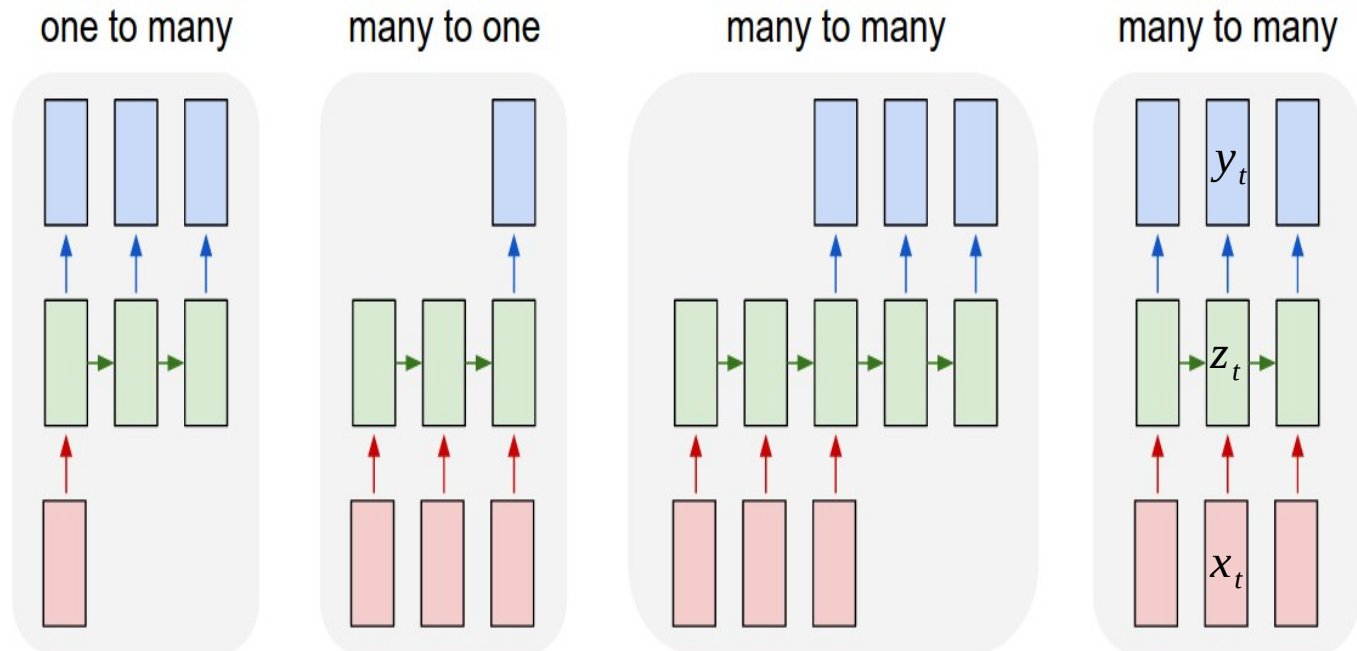
Approximate maximum likelihood sequences

- Approximate maximum likelihood sequence via beam search
 - ▶ Start with empty sequence
 - ▶ Expand previous sequences with all V possible symbols
 - ▶ Compute likelihood of expanded sequences
 - ▶ Keep K best sequences, and proceed to next time step
 - ▶ Terminate after T steps, or when generating STOP symbol
- Computational cost linear in
 - ▶ Beam size K
 - ▶ Vocabulary size V
 - ▶ Sequence length T
- Exhaustive maximum likelihood search exponential in T , since beam size should grow exponentially in that case



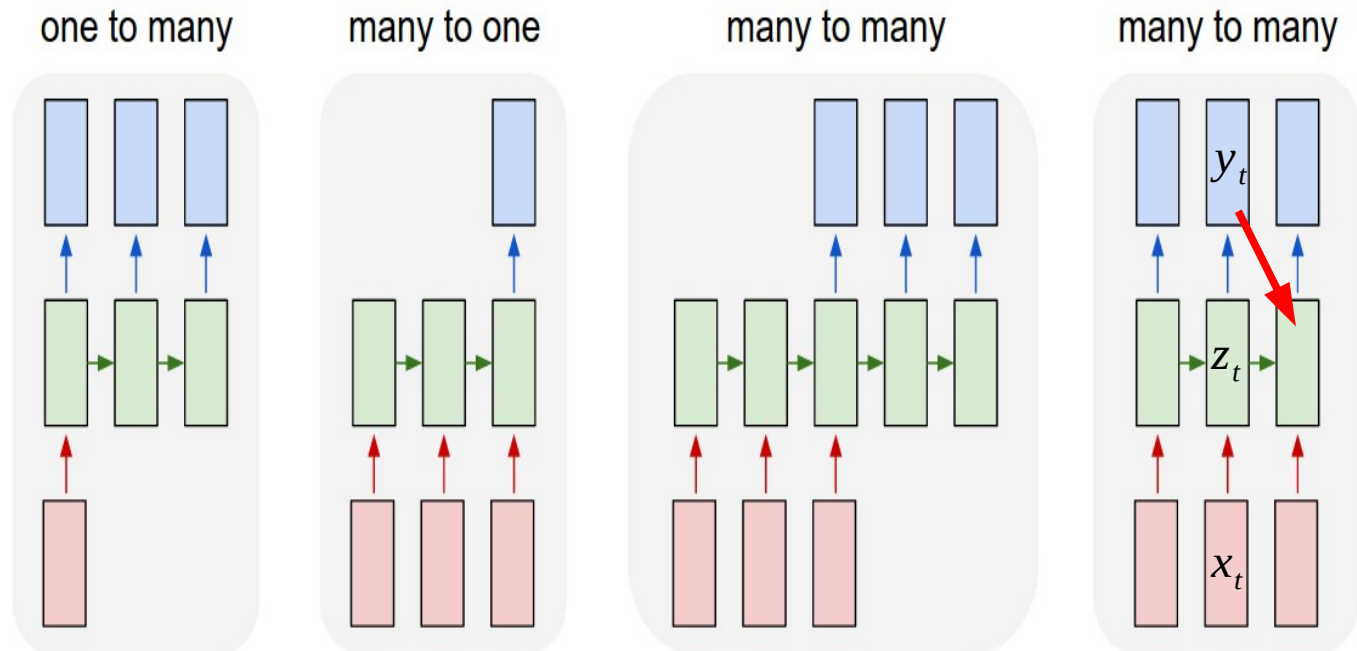
How do we train an RNN ?

- Case without feedback from output
 - ▶ Compute full state sequence given the input (deterministic given input)
 - ▶ Compute loss at each time step wrt ground truth output sequence
 - ▶ Backpropagation through time to compute gradients w.r.t. loss



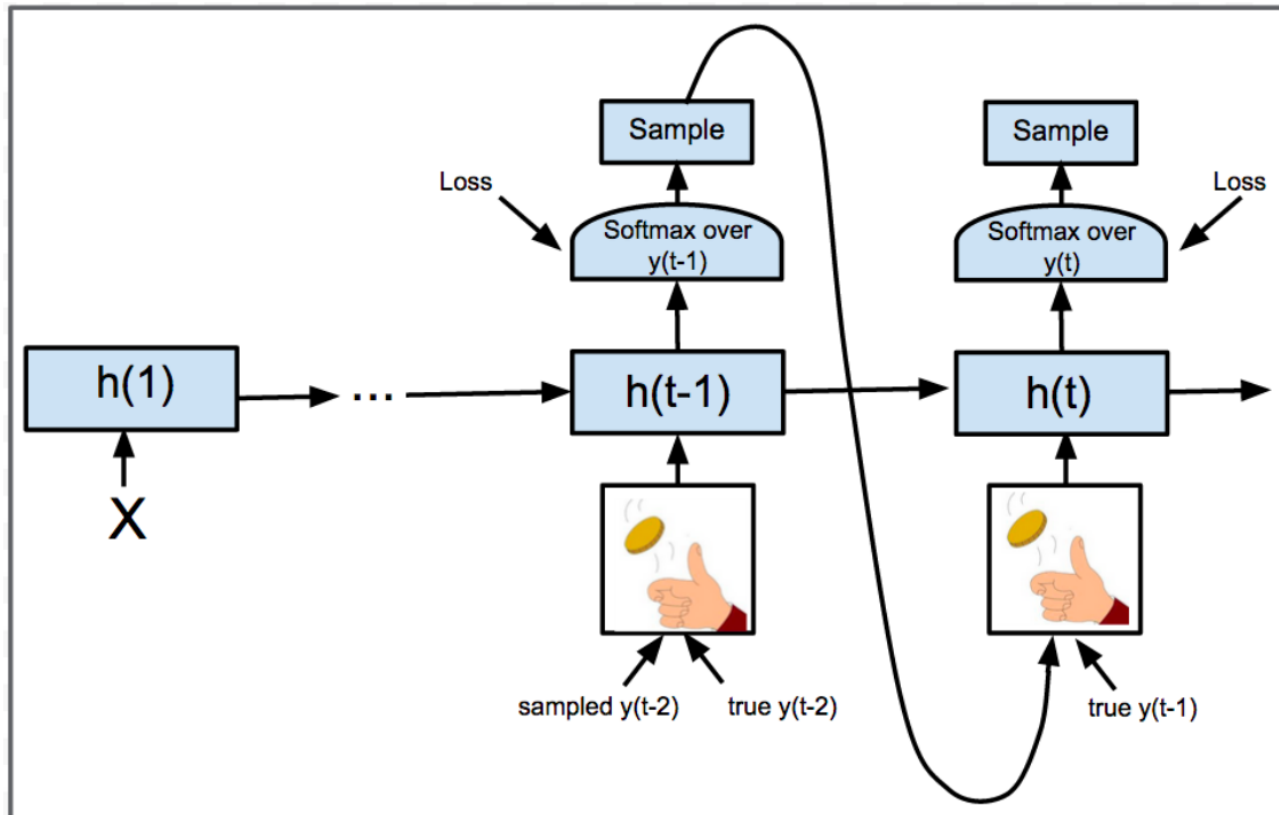
How do we train an RNN ?

- Case with output sequence and feedback
 - ▶ Compute state sequence given input **and ground-truth output**, **deterministic due to known and fixed output**
 - ▶ Loss at each time step wrt ground truth output seq, backprop through time
- Note discrepancy between train and test
 - ▶ Training: predict next symbol correctly given **correct** sequence so far
 - ▶ Test: predict next symbol, given **generated** sequence so far



Scheduled sampling for RNN training

- Compensate discrepancy between train and test procedure by training from generated sequence [Bengio et al. NIPS, 2015]
- Directly training from sampled sequences does not work well in practice
 - ▶ At the start randomly initialized model generates random sequences
 - ▶ Instead, start by training from ground-truth sequence, and progressively increase probability to sample generated symbol in the sequence



Scheduled sampling for RNN training

- Evaluation for image captioning tasks
 - ▶ Higher scores is better
- Dropout as regularization does not help
- Always sampling gives very poor results
- Uniform Scheduled Sampling: sample uniform instead of using model
 - ▶ Already improves over baseline, but not as much as using model

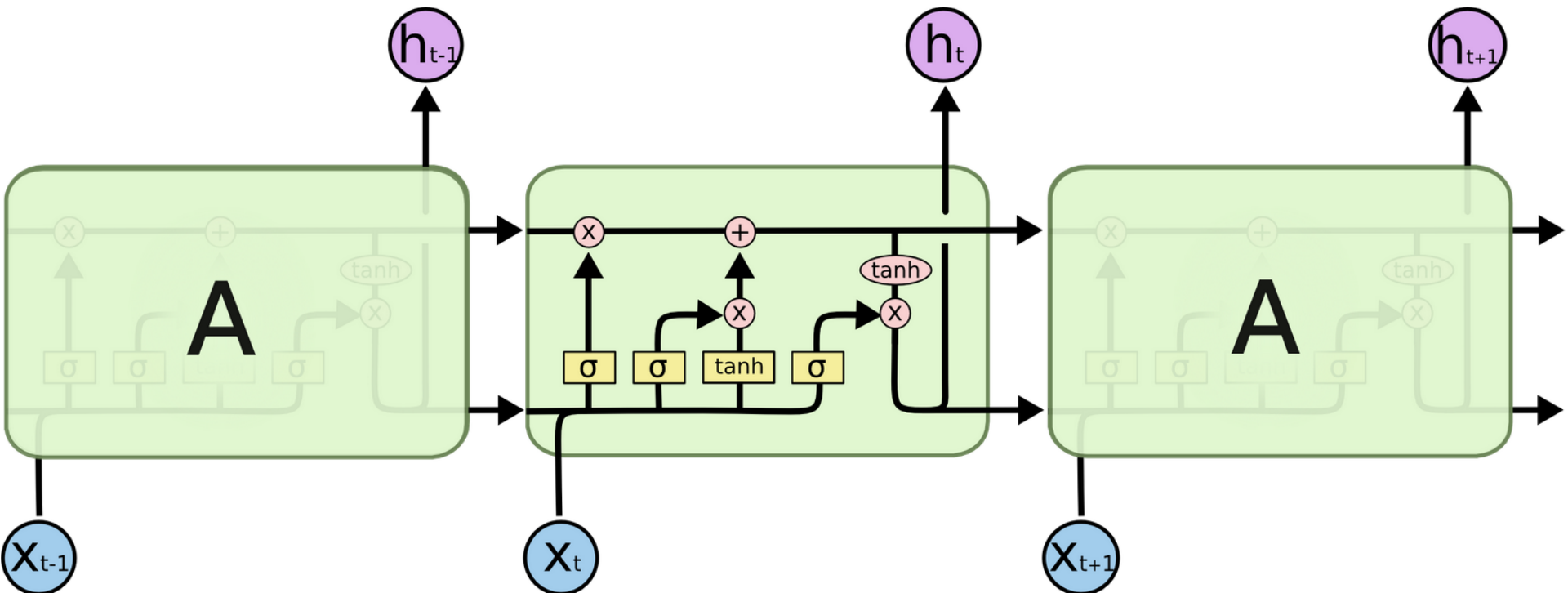
Approach vs Metric	BLEU-4	METEOR	CIDER
Baseline	28.8	24.2	89.5
Baseline with Dropout	28.1	23.9	87.0
Always Sampling	11.2	15.7	49.7
Scheduled Sampling	30.6	24.3	92.1
Uniform Scheduled Sampling	29.2	24.2	90.9
Baseline ensemble of 10	30.7	25.1	95.7
Scheduled Sampling ensemble of 5	32.3	25.4	98.7

Limitations recurrent networks

- Recurrent net can be unrolled as deep network
 - ▶ As deep as the number of time steps of the RNN
 - ▶ Very deep for very long sequences
- Gradients of “deep” layers (far from input) computed via chainrule as product of Jacobians between layers (time-steps)
 - ▶ Product of Jacobians tend to either “explode” to inf. or “vanish” to zero
 - ▶ Similar effect observed in non-recurrent networks
- Approaches to address this issue
 - ▶ Non-recurrent case: add skip connections from earlier layers towards output: Residual networks (He et al, ECCV'16)
 - ▶ Clipping the magnitude of gradients (Pascanu, ICML'13)
 - ▶ Rescaling of gradients based on history of their norm (Kingma & Ba, ICL'15)
 - ▶ Introduction of gates that shield a hidden unit from input and/or output for several layers, effectively shortening the depth for that unit (Hochreiter & Schmidhuber, Neural Computation '97) (Cho et al., Empirical Methods in NLP'14)

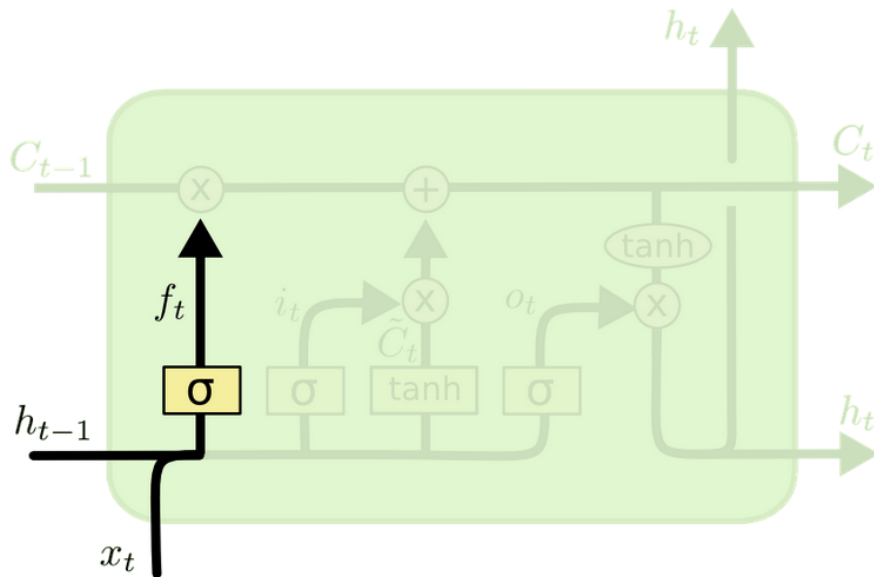
Gated units: Long short-term memory (LSTM)

- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state \mathbf{h} and a “memory cell” \mathbf{c}
- Involves a number of additional processing elements



Long short-term memory (LSTM) cells

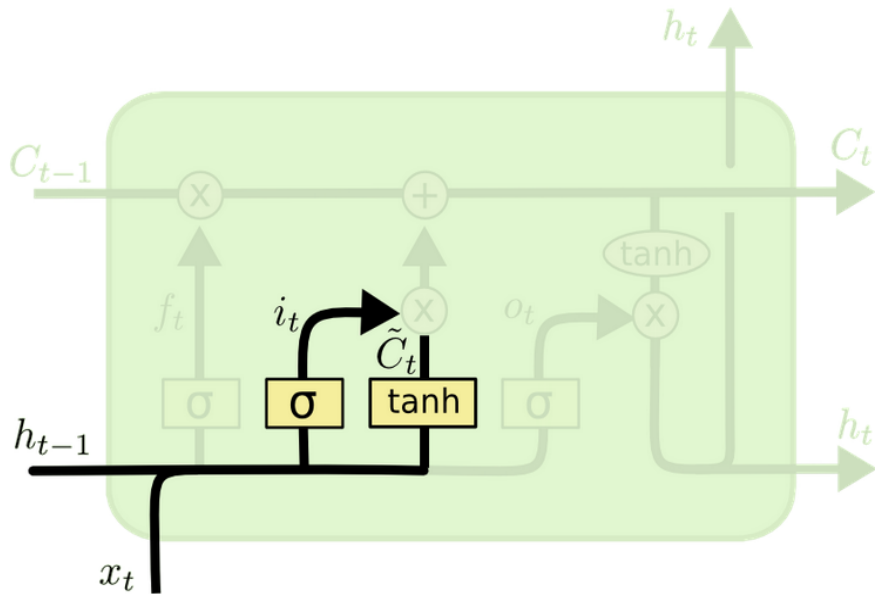
- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state \mathbf{h} and a “memory cell” \mathbf{c}
- Involves a number of additional processing elements
 - ▶ Forget gate \mathbf{f} : “remember” or “forget” previous cell state \mathbf{c}



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Long short-term memory (LSTM) cells

- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state h and a “memory cell” c
- Involves a number of additional processing elements
 - ▶ Input gate i : controls flow of input to cell state
 - ▶ Input modulator \tilde{C} , maps input and previous state to cell state update

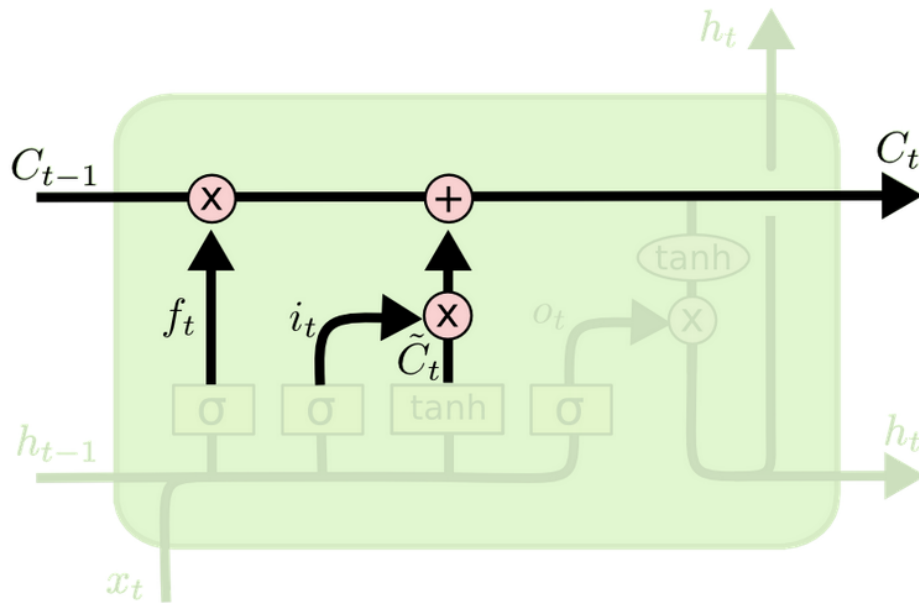


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Long short-term memory (LSTM) cells

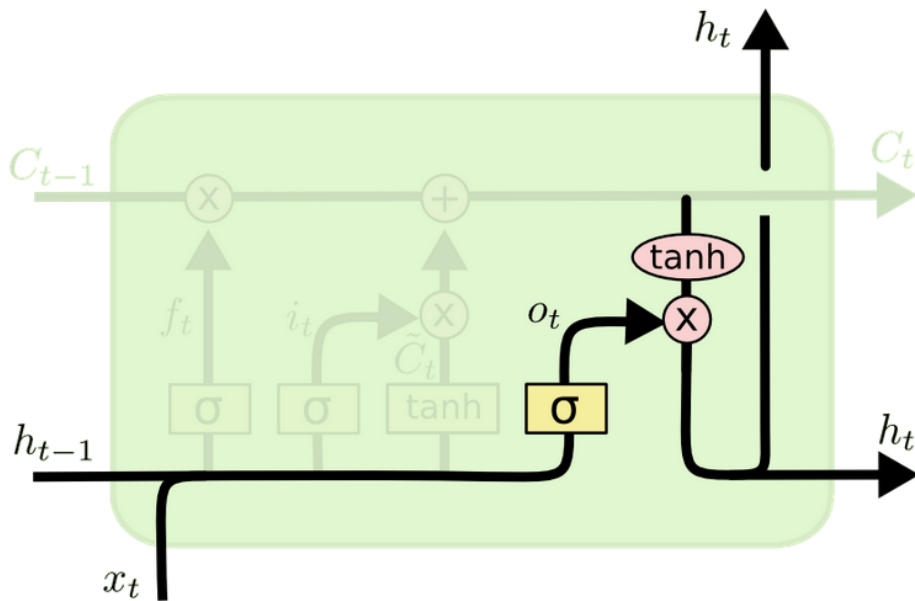
- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state h and a “memory cell” c
- Involves a number of additional processing elements
 - ▶ Cell update: can “forget” previous cell state, can ignore input



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long short-term memory (LSTM) cells

- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state h and a “memory cell” c
- Involves a number of additional processing elements
 - ▶ Output gate o , controls flow of cell state to output
 - ▶ Output vector also passed to next time step of LSTM unit

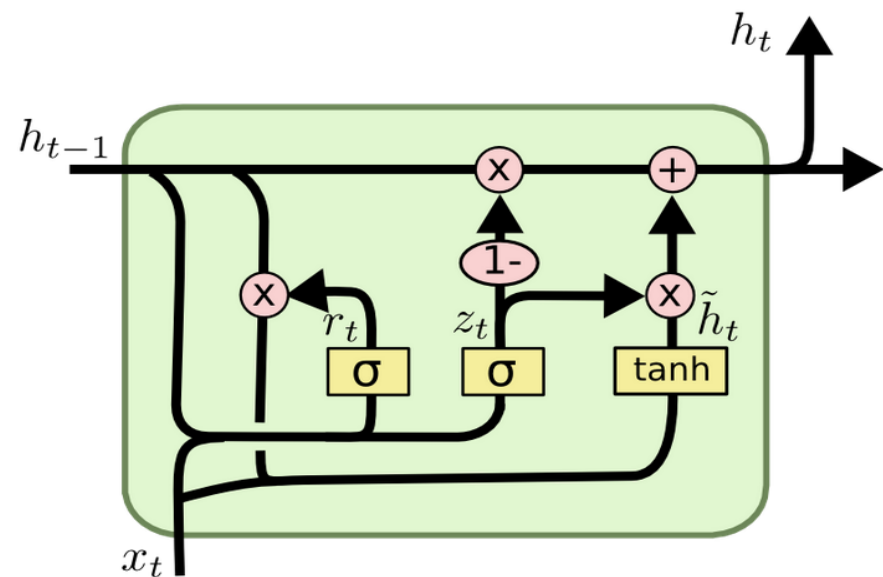


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Gated Recurrent Unit

- GRU simpler gated model than LSTM, empirically gives similar performance (Cho et al., Empirical Methods in Natural Language Processing, 2014)
- Two gates are used
 - ▶ Read gate r , controls access to hidden state for state update
 - ▶ Forget gate z , mixes old state and updated one



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Case study: character-level LSTM language model

- Learn model to generate text at character level without any conditioning
 - ▶ Uses output-feedback for state update
 - ▶ Training data: all Paul Graham essays, about 1 million characters
- Random sample from the trained model:

The surprised in investors weren't going to raise money. I'm not the company with the time there are all interesting quickly, don't have to get off the same programmers. There's a super-angel round fundraising, why do you can do. If you have a different physical investment are become in people who reduced in a startup with the way to argument the acquirer could see them just that you're also the founders will part of users' affords that and an alternation to the idea. [2] Don't work at first member to see the way kids will seem in advance of a bad successful startup. And if you have to act the big company too.

- Model learned to spell words, and long-range grammatical dependencies

Case study: character-level LSTM language model

- Same model, now training data: all of Shakespeare (4.4 MB)
- Random sample from the trained model:

PANDARUS:

*Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.*

Second Senator:

*They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.*

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

*They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.*

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

- Specific style structure is also captured by the model

Case study: character-level LSTM language model

- Same model trained on linux source code (474 MB)
 - ▶ Very long range dependencies on bracket structure

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
    return segtable;
}
```


Case study: image captioning

- Given image generate descriptive english sentence



a brown dog is running through the grass

Image captioning with encoder-decode system

- Encoder: CNN takes image and maps it into a vector
- For example, fully connected layer of VGG16 network
 - ▶ CNN pre-trained on ImageNet classification task (>1 million images)

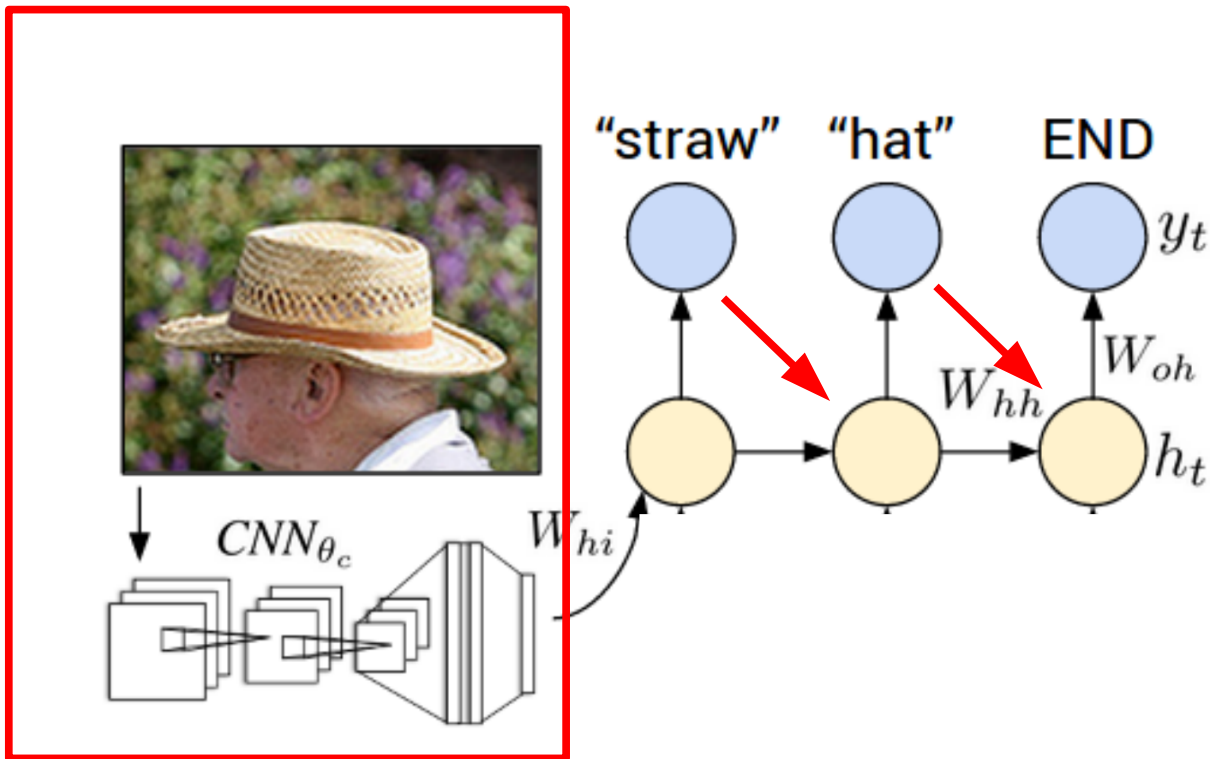


Image captioning with encoder-decode system

- Decoder: RNN takes CNN image vector to initialize RNN state
- Typical configuration:
 - ▶ Single layer of 512 GRUs
 - ▶ Output feedback to ensure coherent sentence

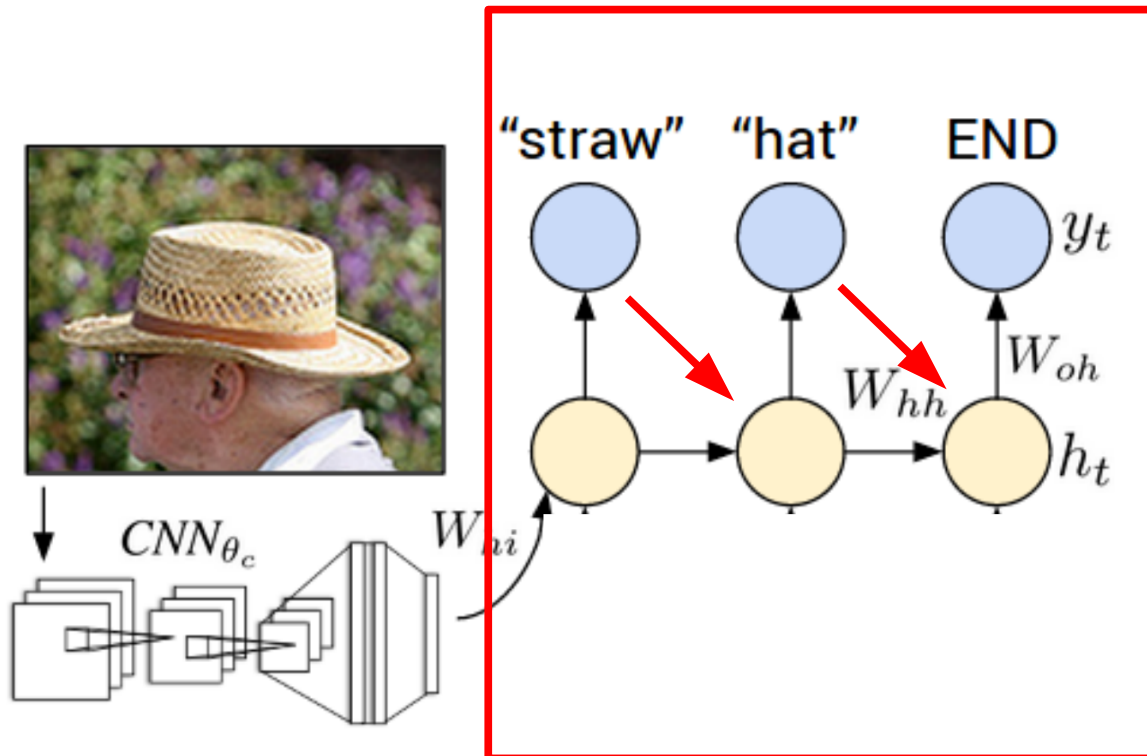


Image captioning with encoder-decode system

- Example output (Vinyals et al., CVPR 2015)

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A group of young people playing a game of frisbee.

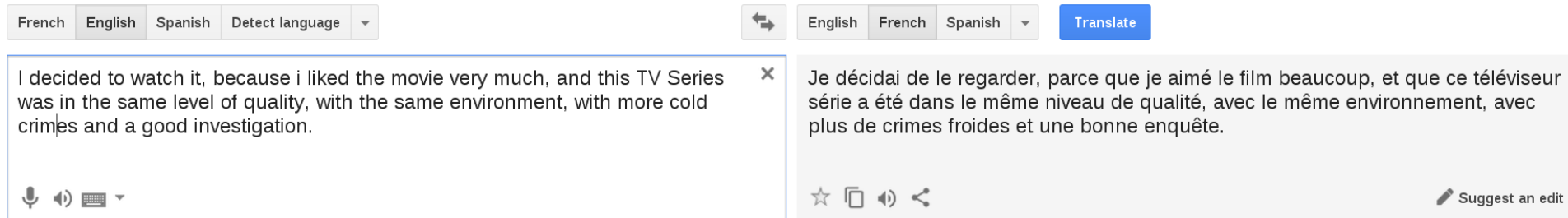


Two hockey players are fighting over the puck.



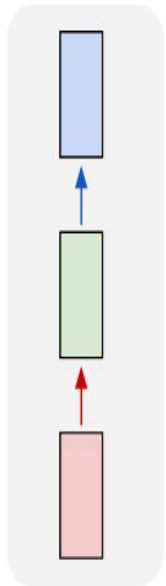
Case-study: Machine translation with encoder-decoder

- Translation of a sentence into another language
 - ▶ Input and output of different length

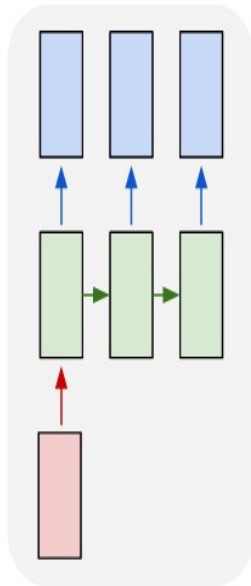


The screenshot shows a web-based machine translation interface. On the left, there are language selection buttons for French, English, and Spanish, and a 'Detect language' dropdown. The input text is: "I decided to watch it, because i liked the movie very much, and this TV Series was in the same level of quality, with the same environment, with more cold crimes and a good investigation." On the right, the output text is: "Je décidai de le regarder, parce que je aimé le film beaucoup, et que ce téléviseur série a été dans le même niveau de qualité, avec le même environnement, avec plus de crimes froides et une bonne enquête." A 'Translate' button is visible between the input and output fields.

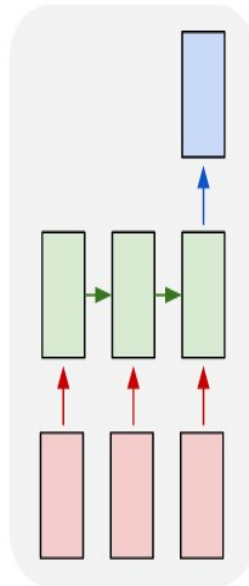
one to one



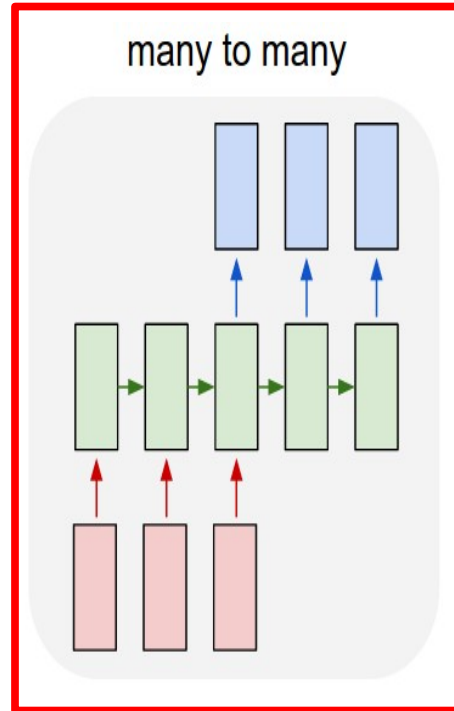
one to many



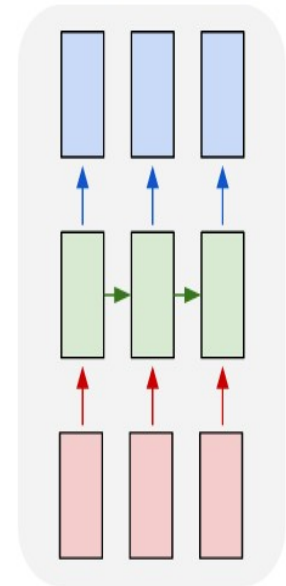
many to one



many to many

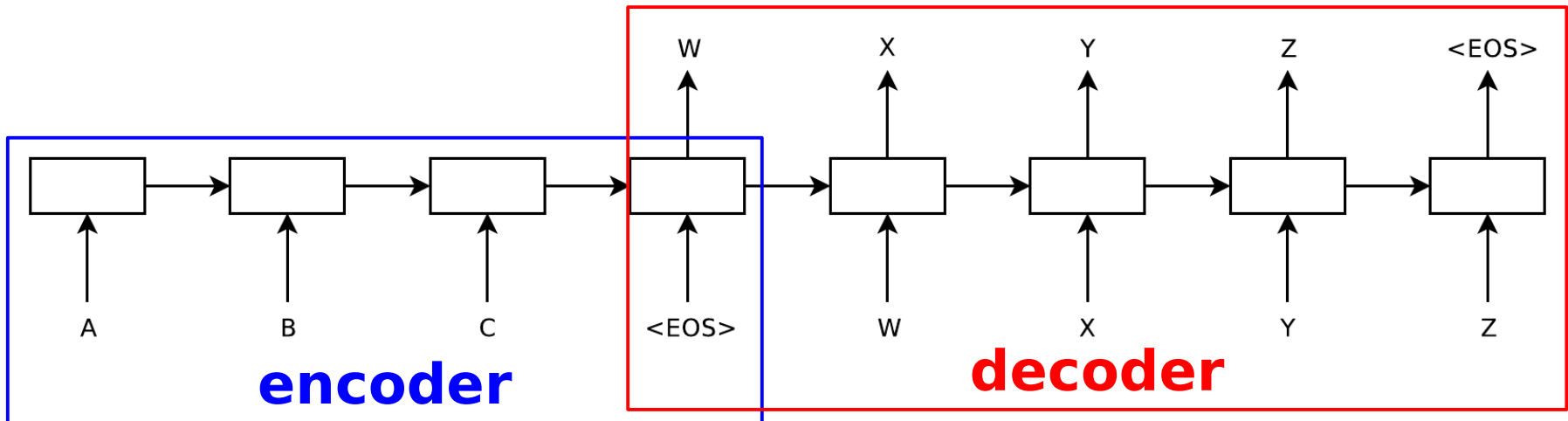


many to many



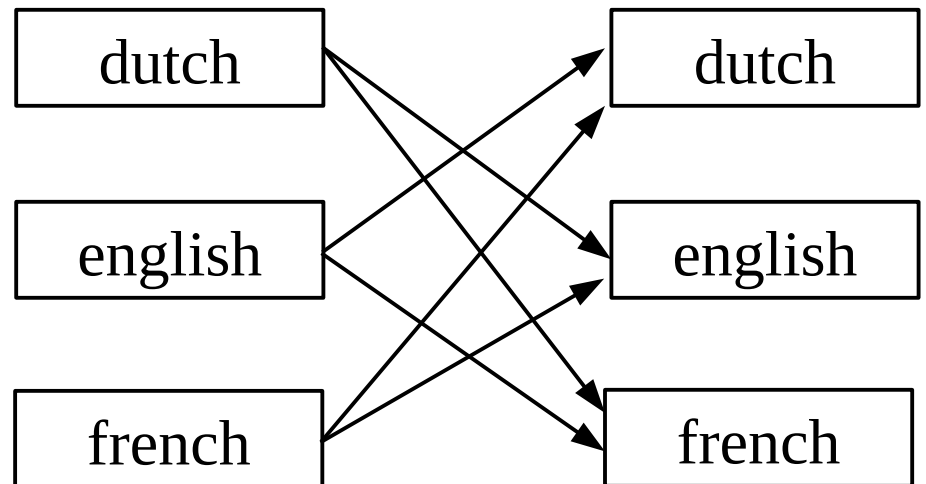
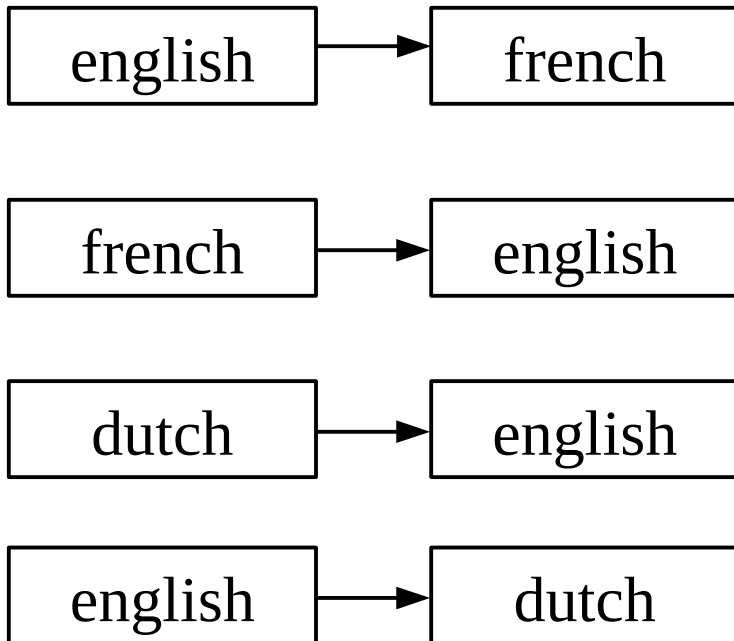
Encoder-decoder model

- Read source sentence with **encoder** RNN (Sutskever et al., NIPS 2014)
 - ▶ Reading input in reverse yields better result: more short dependencies
 - ▶ Can use bidirectional RNN since input sequence is given
- Generate target sentence with **decoder** RNN
 - ▶ Uses a different set of parameters
 - ▶ Uses output feedback to ensure output coherency
- Meaning of source sentence encoded in the RNN state vector passed between encoder and decoder
 - ▶ The captioning model we saw before is a variant of this idea where a CNN is used as an image encoder



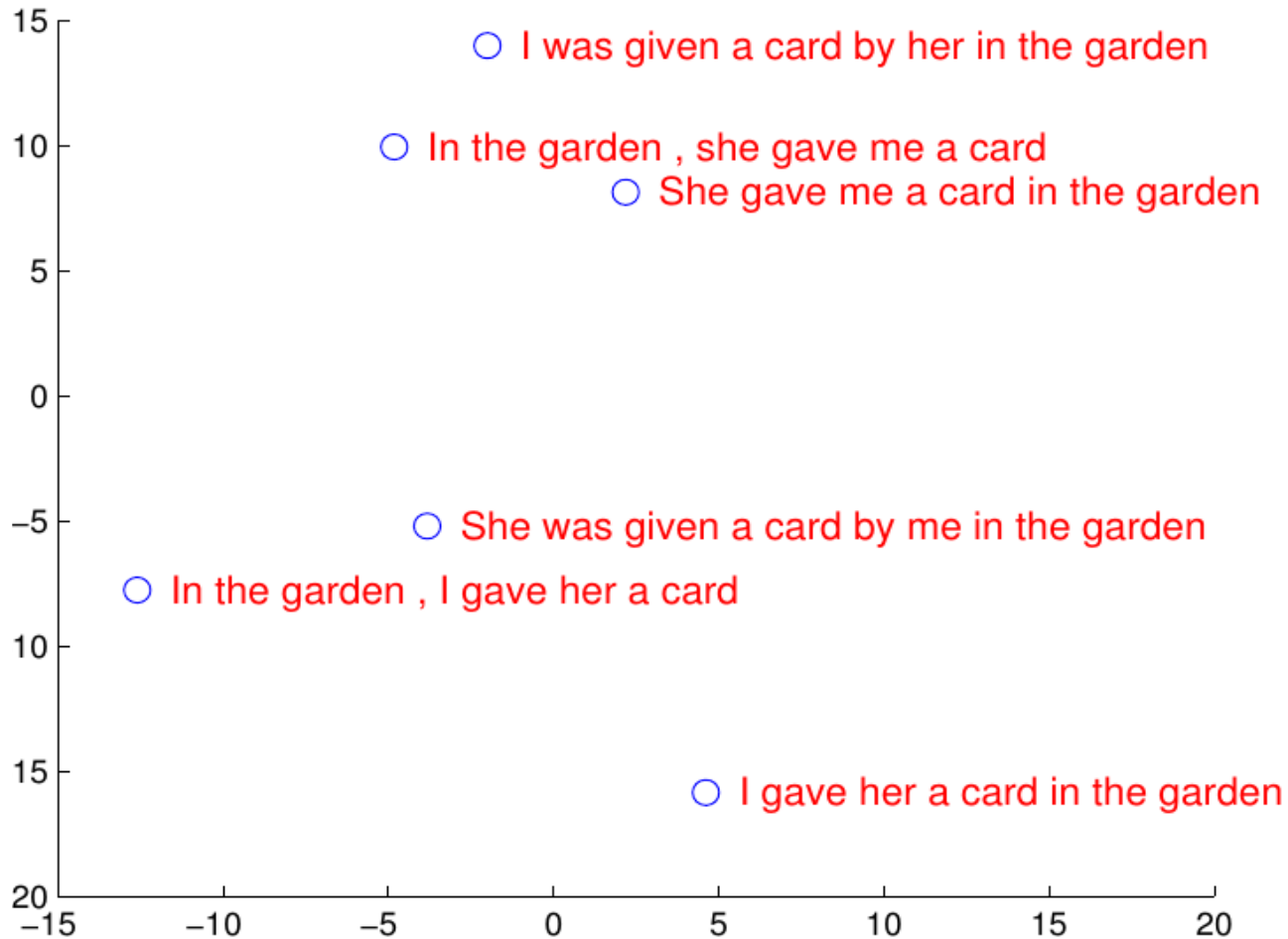
Encoder-decoder machine translation

- Trained from “aligned” corpus of matching source-target sentences
- **Encoder** and **decoder** can be learned on multiple language pairs in parallel
 - ▶ (English to French) and (Dutch to French) use same decoder
 - ▶ (English to French) and (English to Dutch) use same encoder
- Generalizes to translation between new language pairs for which no aligned training corpus was available



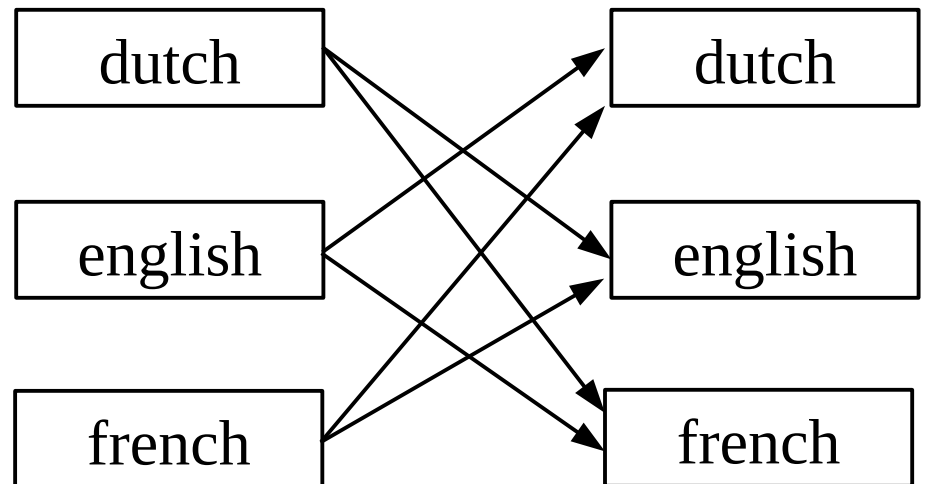
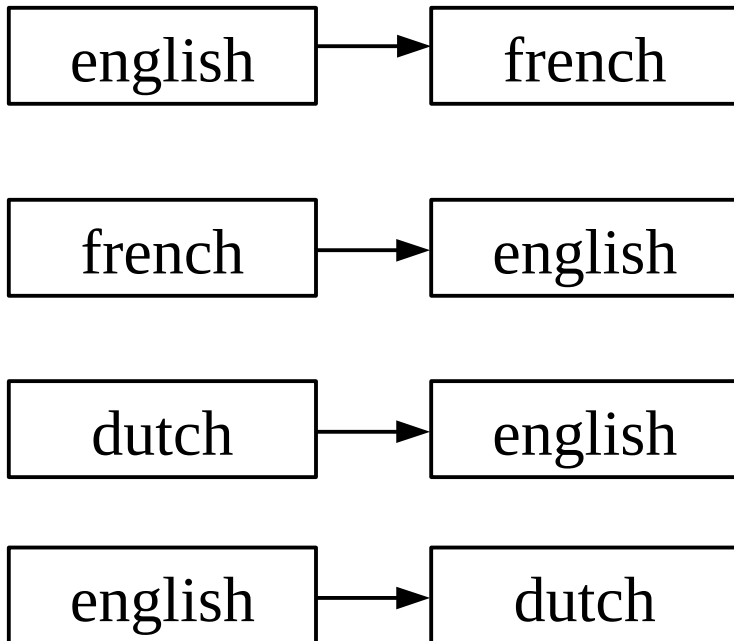
Encoder-decoder machine translation

- PCA projection of LSTM encoder state after reading several phrases
- Word order important for meaning, captured in encoder state vector



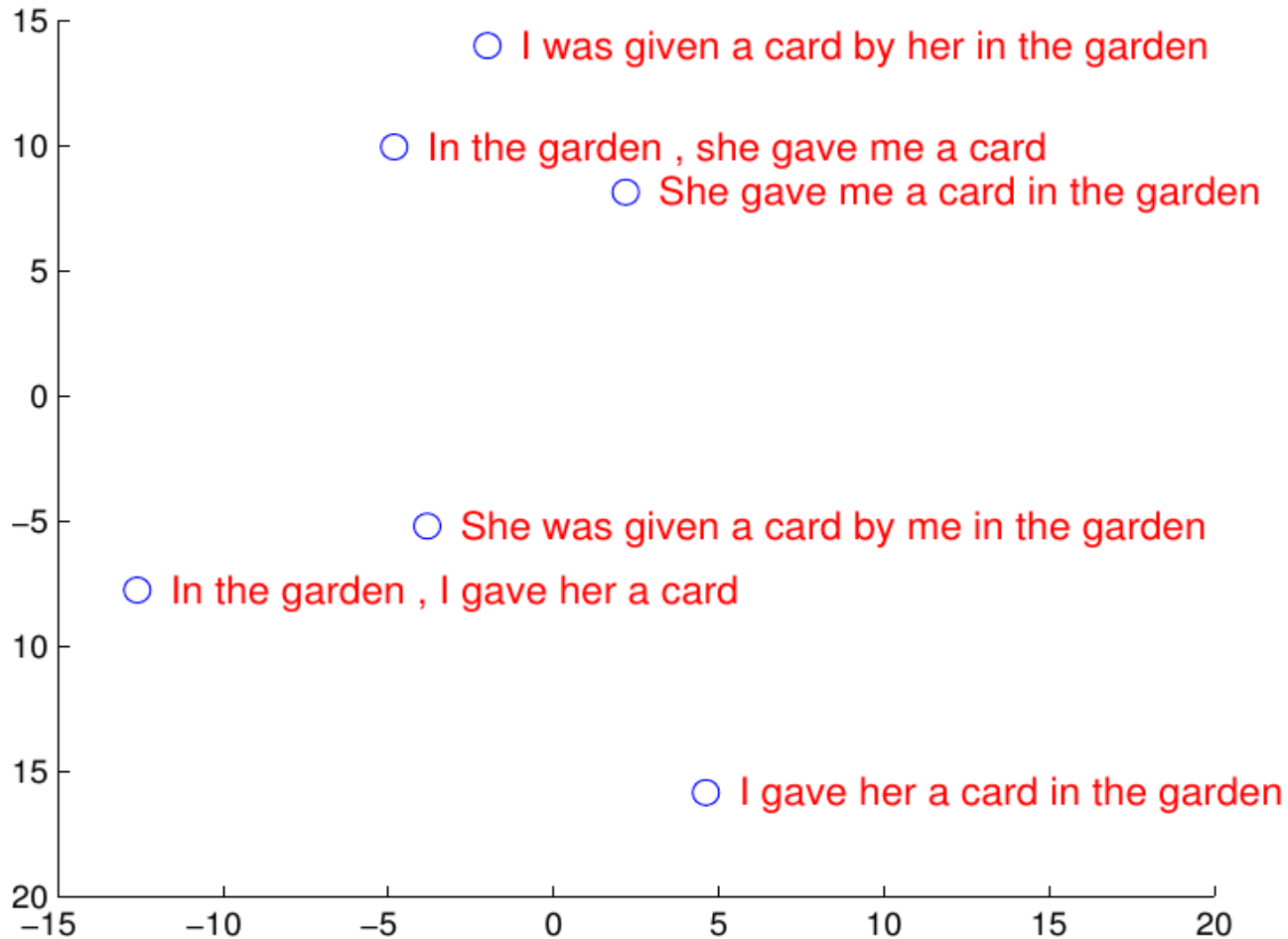
Encoder-decoder machine translation

- Trained from “aligned” corpus of matching source-target sentences
- **Encoder** and **decoder** can be learned on multiple language pairs in parallel
 - ▶ (English to French) and (Dutch to French) use same decoder
 - ▶ (English to French) and (English to Dutch) use same encoder
- Generalizes to translation between new language pairs for which no aligned training corpus was available



Encoder-decoder machine translation

- PCA projection of LSTM encoder state after reading several phrases
- Word order important for meaning, captured in encoder state vector



Summary of recurrent networks

- Recurrent networks are powerful tools to model sequential data
 - ▶ Sequential input and/or sequential output
 - ▶ Input and output sequence may be aligned or not
- “Unfolding” recurrent networks allows to recognize them as feedforward networks with weight-sharing across recurrent connections, generalizable to
 - ▶ Multidimensional structures data on input or output
 - ▶ Encoder-decoder networks
 - ▶ Deep recurrent networks
- Recurrent networks with a gating mechanism much more powerful for tasks with long-range dependencies: LSTM and GRU
 - ▶ Gates can be thought of as internal to the recurrent unit
 - ▶ Does not change the unfolded graph topology: remains feedforward
- State-of-the art performance on various tasks, including
 - ▶ Image captioning
 - ▶ Machine translation
 - ▶ Speech recognition

Further reading

- “Deep Learning”
Ian Goodfellow, Yoshua Bengio, Aaron Courville.
MIT Press, 2016 (free online)
<http://www.deeplearningbook.org/>
- “Supervised Sequence Labelling with Recurrent Neural Networks”
Alex Graves, 2012 (free online)
- “Pattern Recognition and Machine Learning”
Chris Bishop.
Springer, 2006.