Introduction to Neural Networks

Advanced Learning Models 2015-2016

Jakob Verbeek, January 21+28, 2016

Course website:

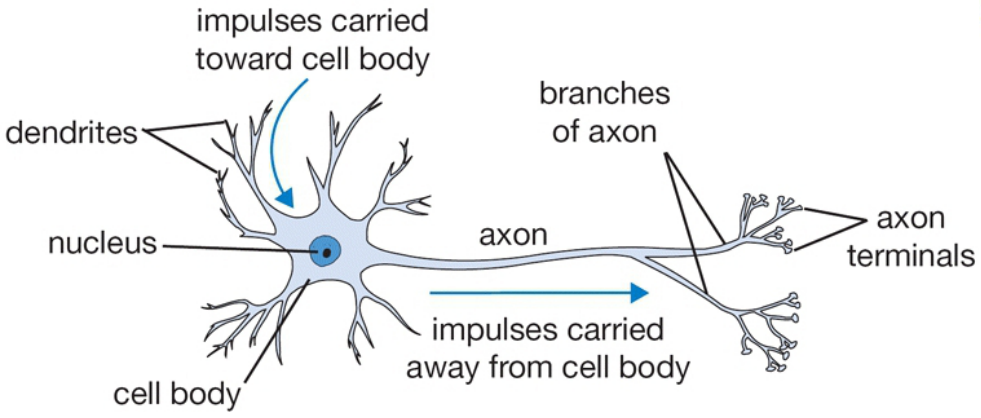
http://lear.inrialpes.fr/people/mairal/teaching/2015-2016/MSIAM

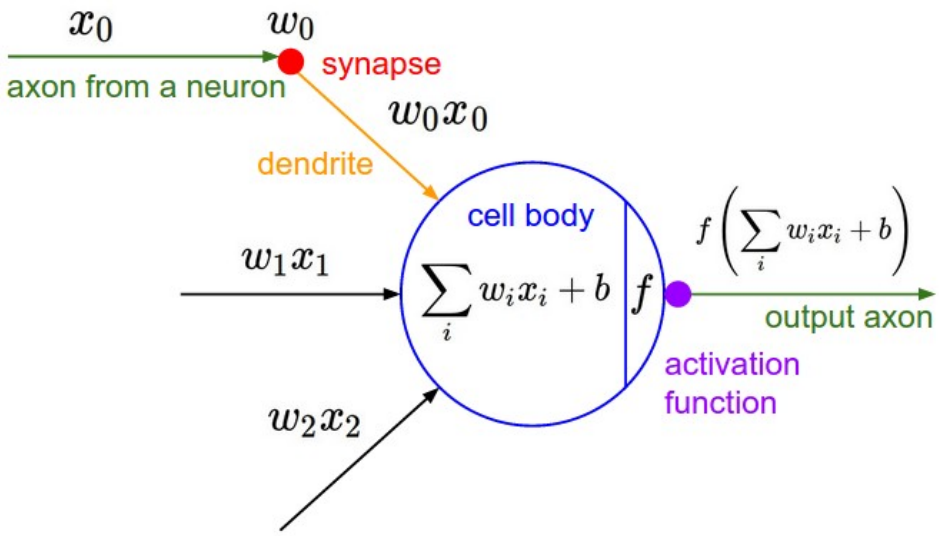
Kaggle data challenge

- https://inclass.kaggle.com/c/advanced-learning-models-msiam
- After the deadline of the data challenge, you have to send by email
  - a small report on what you did (in pdf format, 11pt, 2 pages A4 max)
  - your source code (zip archive), with a simple script "start" (that may be called from Matlab, Python or R) which will reproduce your submission
- Rules
  - At most 2 persons per team.
  - You can submit results up to twice per day during the challenge.
  - A leader board will be available during the challenge, which shows the best results per team, as measured on a subset of the test set.
  - A different part of the test set will be used after the challenge to evaluate the results.
  - Registration has to be done with email addresses @grenoble-inp.fr or @e.ujf-grenoble.fr
- **The most important rule is: DO IT YOURSELF.** The goal is to gain practical experience with the machine learning techniques involved.
  - For this reason, the use of external machine learning libraries is forbidden. For instance, this includes, but is not limited to, libsvm, liblinear, scikit-learn, ...
  - On the other hand, you are welcome to use general purpose libraries, such as for linear algebra or optimization

Biological motivation

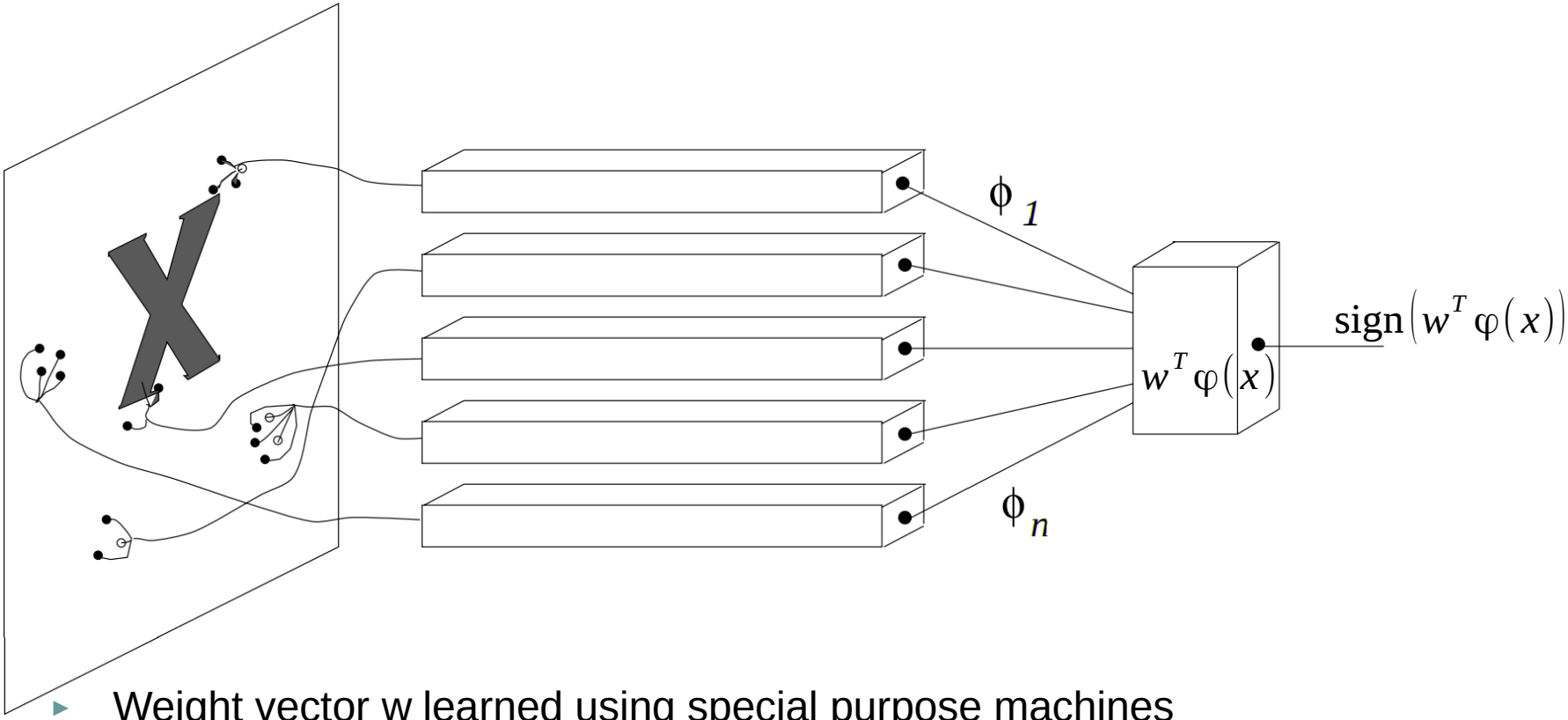
- Neuron is basic computational unit of the brain
  - about 10^11 neurons in human brain
- Simplified neuron model as linear threshold unit (McCulloch & Pitts, 1943)
  - Firing rate of electrical spikes modeled as continuous output quantity
  - Multiplicative interaction of input and connection strength (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

- One of the earliest works on artificial neural networks
  - First implementations in 1957 at Cornell University
  - Computational model of natural neural learning



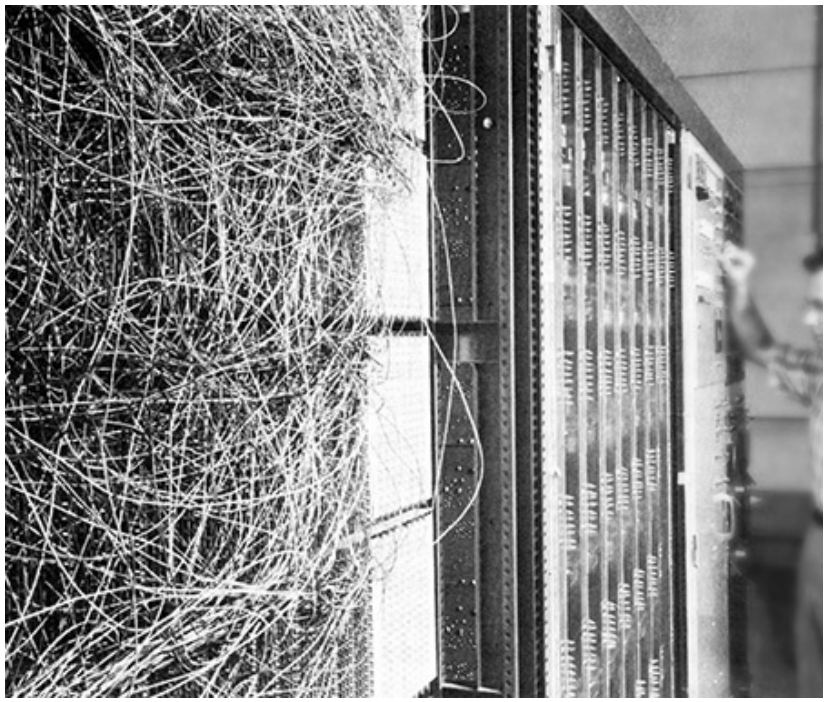
  - Weight vector w learned using special purpose machines
  - Associative units fixed by lack of learning rule at the time

Rosenblatt's Perceptron

- One of the earliest works on artificial neural networks
  - First implementations in 1957 at Cornell University
  - Computational model of natural neural learning
- Binary classification based on sign of generalized linear function

$$sign(f(x)) = sign(w^T \varphi(x))$$

20x20 pixel sensor



Random wiring of associative units

Rosenblatt's Perceptron

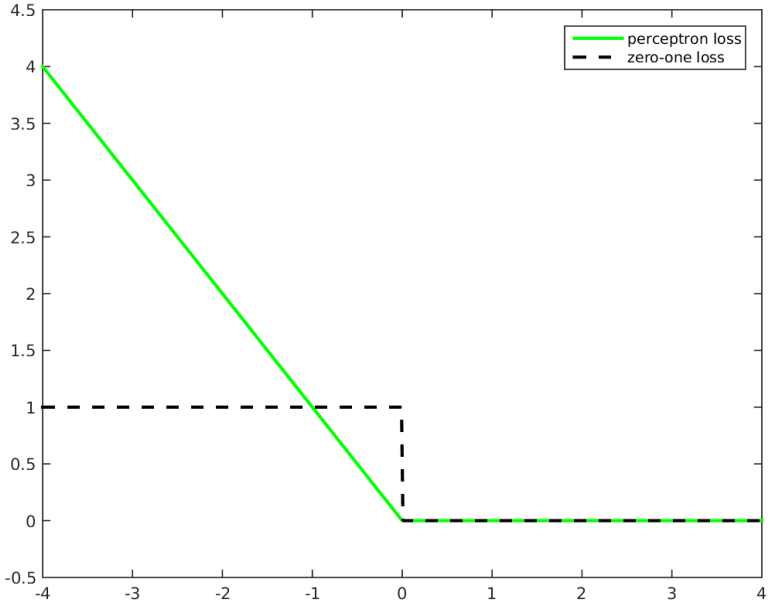
- Objective function linear in score over misclassified patterns $t_i \in \{-1, +1\}$

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

- Perceptron learning via stochastic gradient descent

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

  - 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

$$\begin{aligned}\langle w', w'\rangle &= \langle w, w\rangle + 2\, y\langle w, x\rangle + \langle x, x\rangle \\ &< \langle w, w\rangle + \langle x, x\rangle \\ &< \langle w, w\rangle + M\end{aligned}$$

- Thus after t updates we have

$$\langle w^*, w'\rangle > \langle w^*, w\rangle + t\,\delta$$
$$\langle w', w'\rangle < \langle w, w\rangle + tM$$

and therefore $a(t) = \frac{\langle w^*, w(t)\rangle}{\langle w(t), w(t)\rangle} > \frac{\langle w^*, w\rangle + t\,\delta}{\sqrt{\langle w, w\rangle + tM}}$, 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 $t \le \frac{M}{\delta^2}$

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)

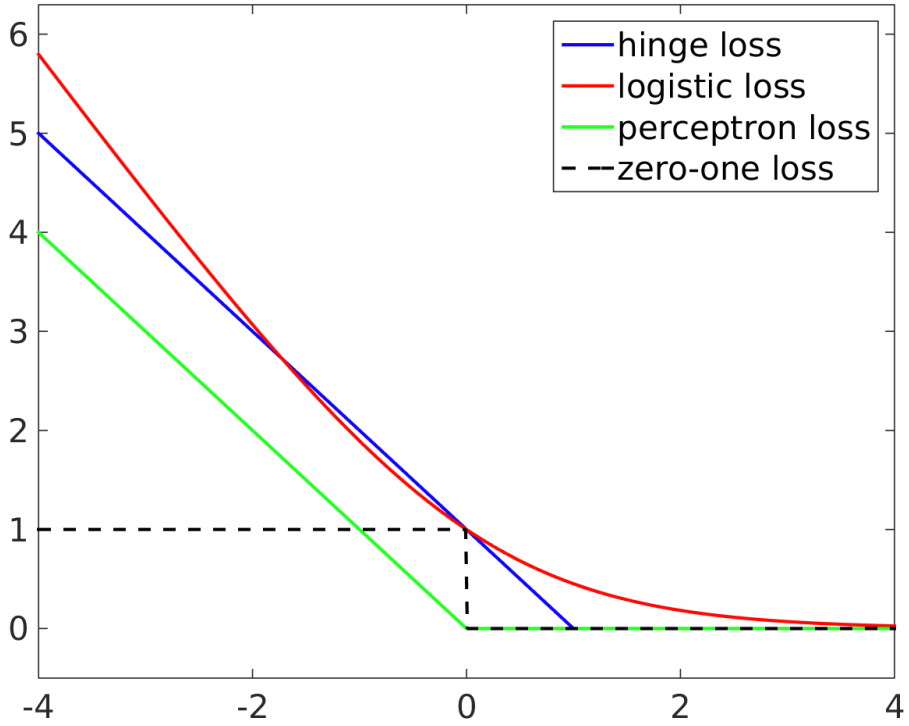
- If training data is linearly separable then the found solution will depend on the initialization and ordering of data in the updates
- If training data is not linearly separable, then the perceptron learning algorithm will not converge

- No direct multi-class extension
- No probabilistic output or confidence on classification

Relation to SVM and logistic regression

- Perceptron similar to SVM 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

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



Kernels to go beyond linear classification

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

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

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

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

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

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

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

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 $\varphi(x)$
  - 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

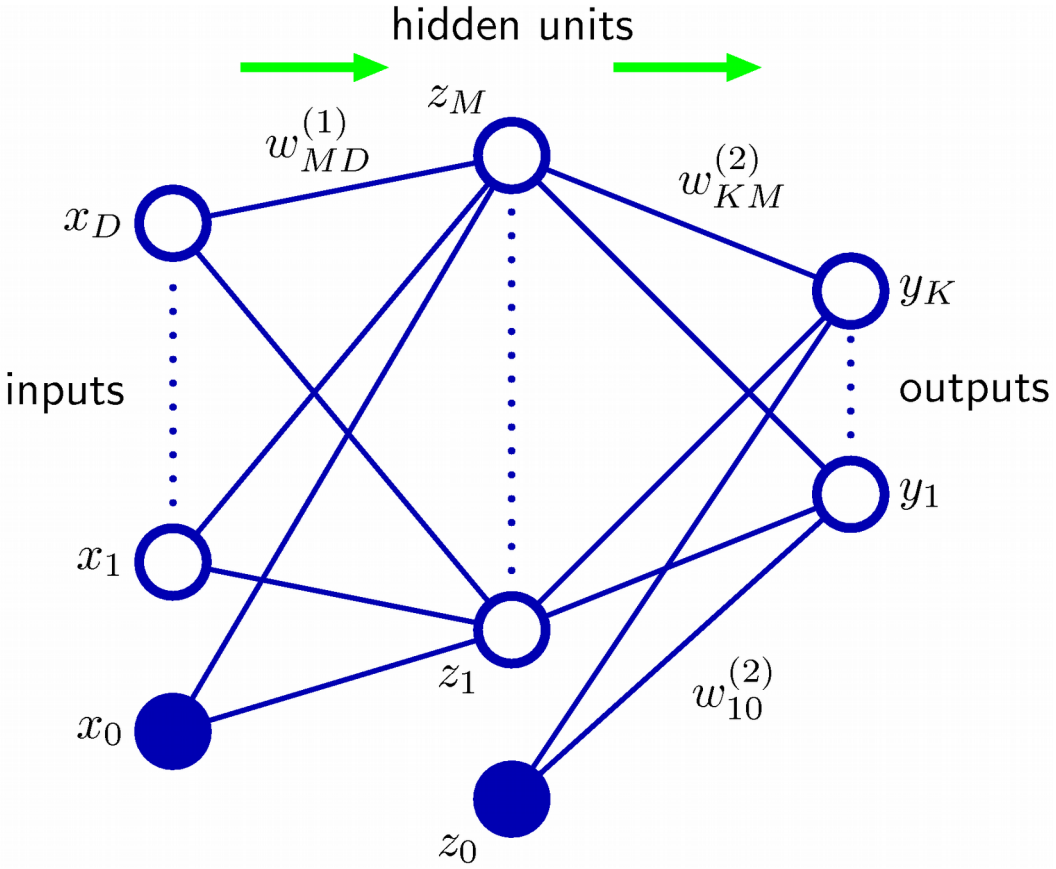
$$f(x) = \sum_i \alpha_i \varphi_i(x; \theta_i)$$

Feed-forward neural networks

- Define outputs of one layer as scalar non-linearity of linear function of input
- Known as “multi-layer perceptron”
  - Perceptron has a step non-linearity of linear function
  - Typically other non-linearities are used in practice

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

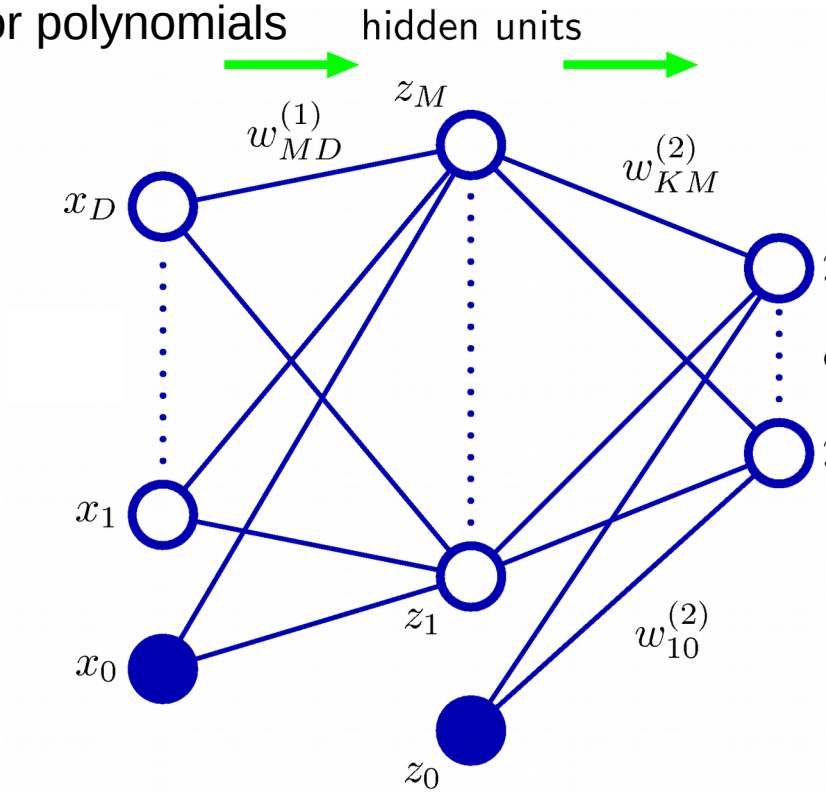
$$y_k = \sigma\left(\sum_j z_j w_{jk}^{(2)}\right)$$



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



Classification over binary inputs

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

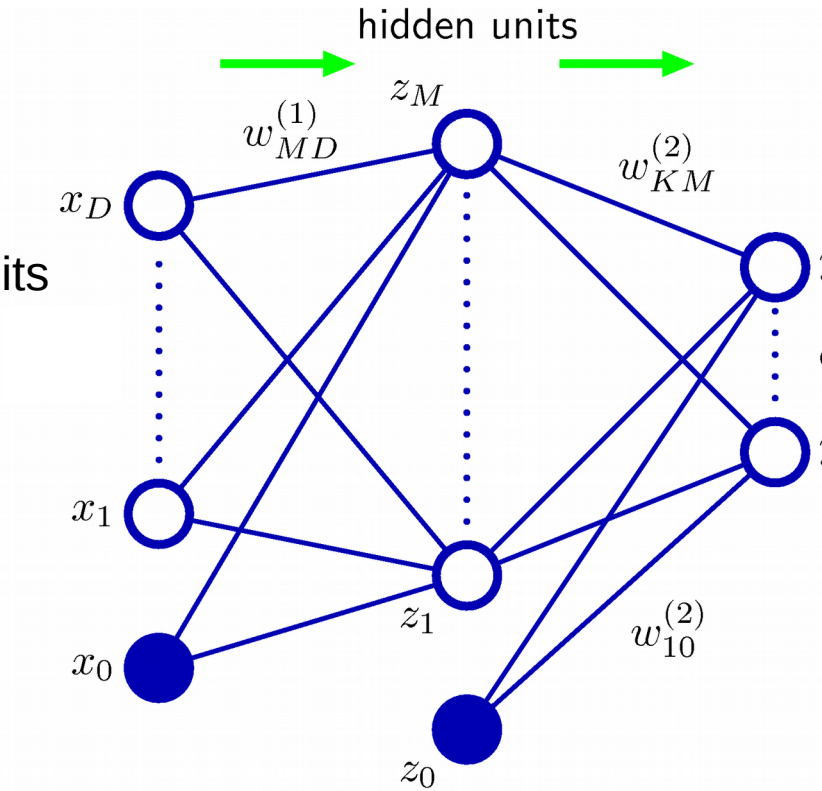
$$z_m = \text{sign}\left(\sum_{i=1}^{D} w_{mi} x_i - D + 1\right)$$

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

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

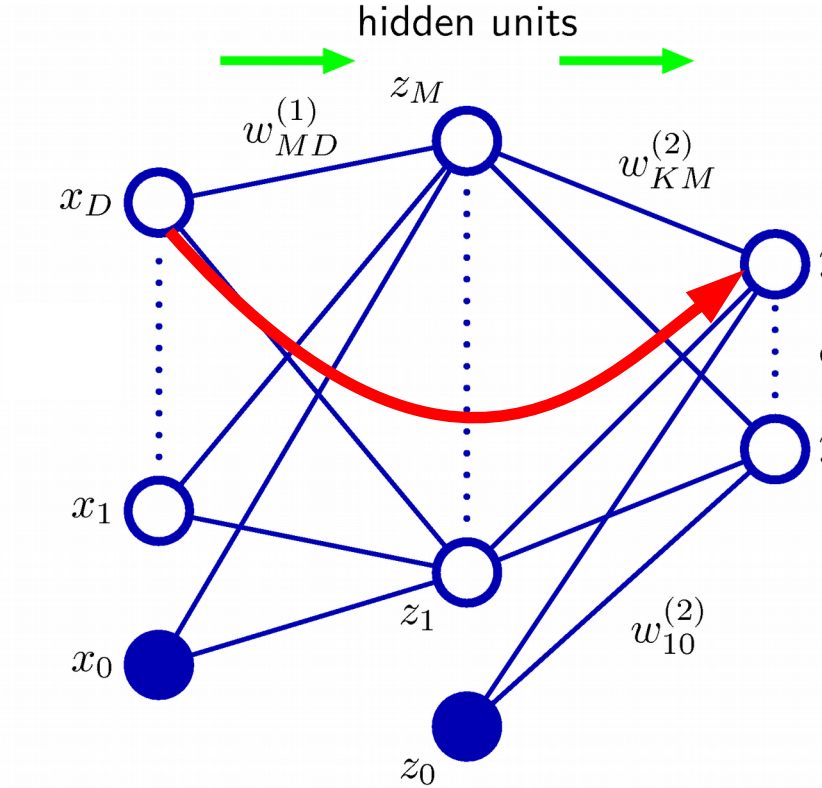
$$y = \text{sign}\left(\sum_{m=1}^{M} z_m + M - 1\right)$$

- 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:
  - Network architecture design
    - Nr nodes, layers, non-linearities,
  - Learn the optimal parameters
    - Non-convex optimization



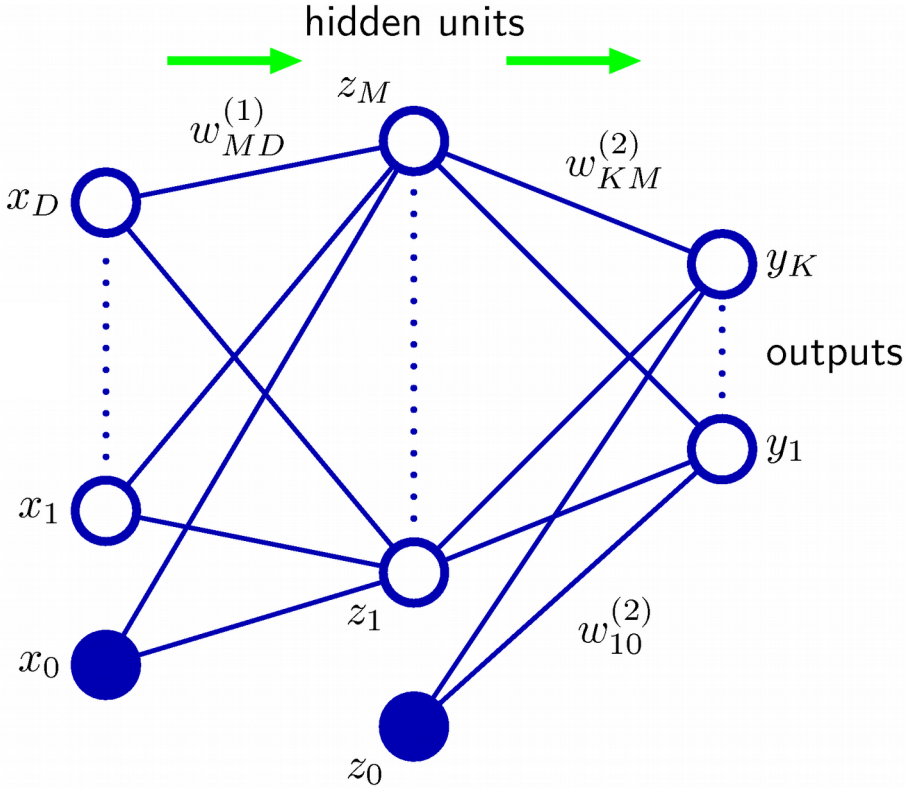
An example: multi-class classifiction

- 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

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

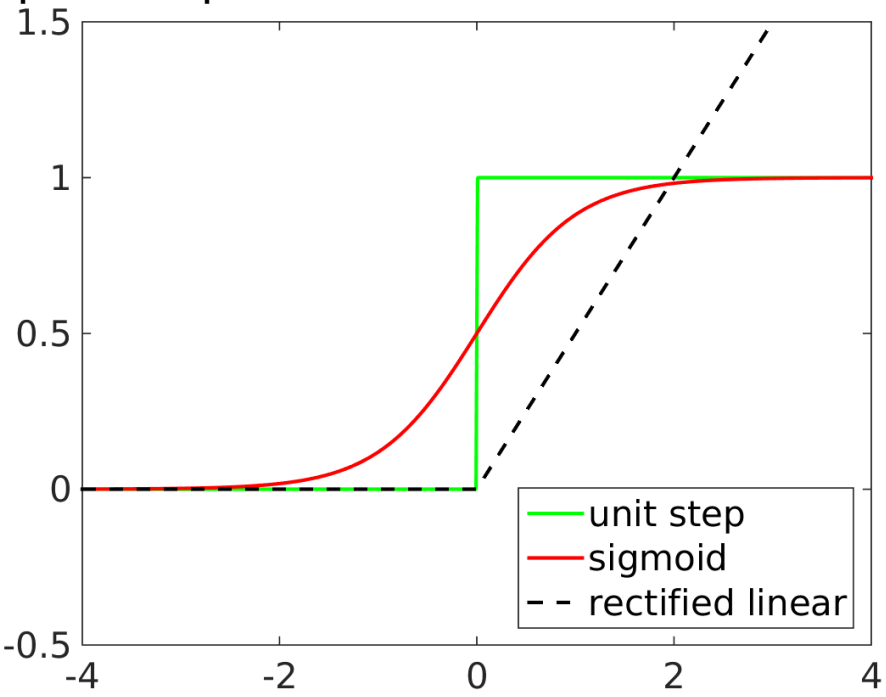
- Precisely as before, only difference is that we are now learning the data transformation concurrently with the classifier

- 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

- Unit step function, used in original Perceptron
  - Discontinuous, not possible to propagate error
- Sigmoid function: Smooth step function
  - Gradients saturate except in transition regime
  - Hyperbolic tangent: same but zero-centered instead
- Rectified linear unit (ReLU): Clips negative values to zero
  - One-sided saturation only, very cheap to compute
- Max-out: max of two linear functions
  - Similar as ReLU
  - No constant regimes at all



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

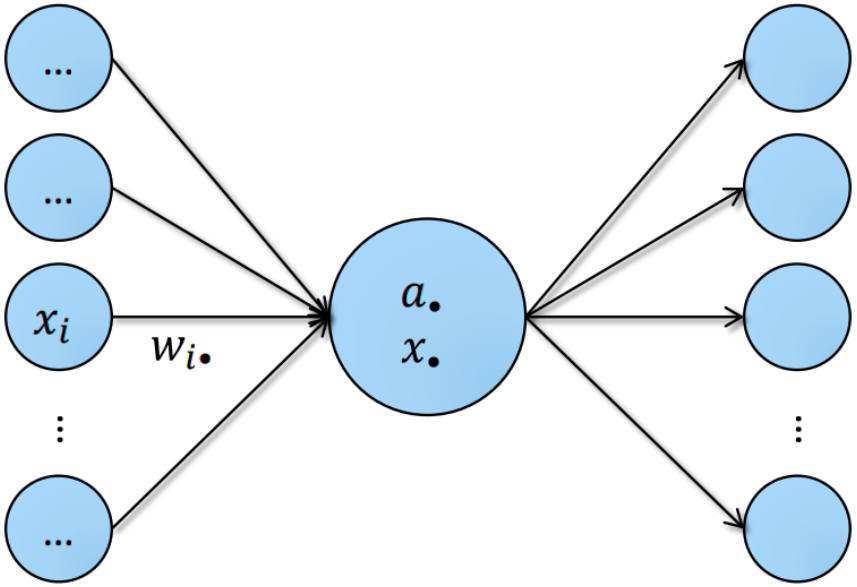
$$\frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i; W\right) + \lambda\,\Omega\left(W\right)$$

- 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 Pre(j) to denote all nodes feeding into j



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

$$x_j = f(a_j)$$

Training the network: backward propagation

- Input aggregation and activation

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

$$x_j = f(a_j)$$

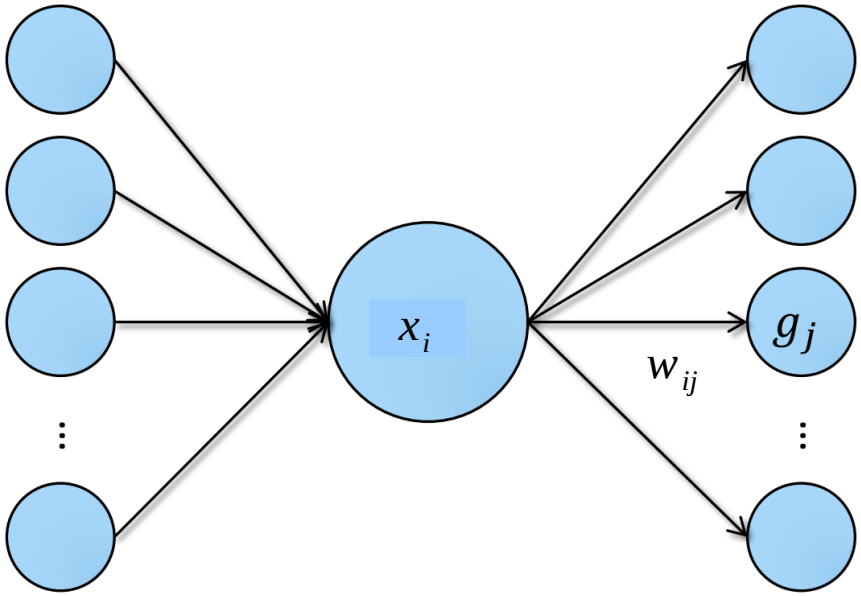
- Partial derivative of loss w.r.t. input

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

- 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
  - Matrix-vector product
- Multiply with derivative of local activation
  - Point-wise multiplications

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

$$x_j = f(a_j)$$

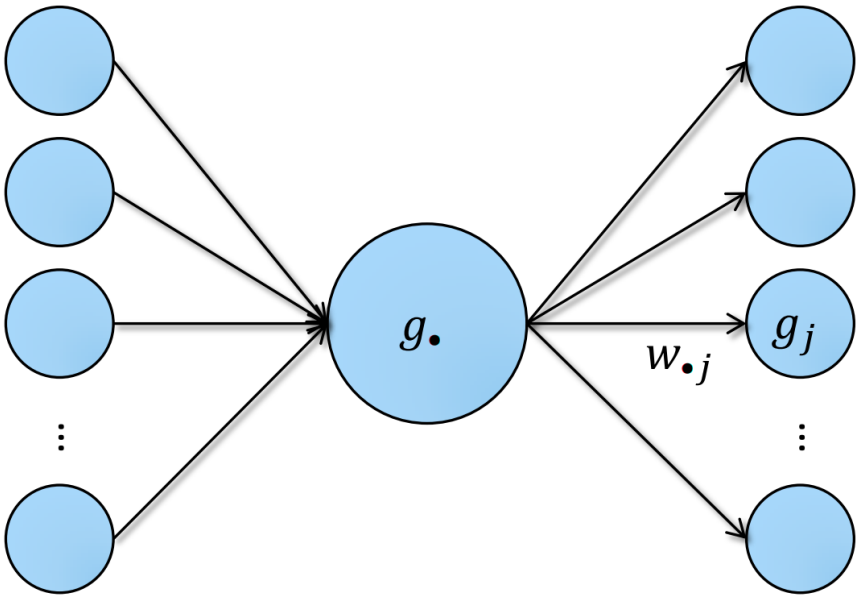
$$g_i = \frac{\partial L}{\partial a_i}$$

$$\frac{\partial L}{\partial x_i} = \sum_{j \in Post(i)} \frac{\partial a_j}{\partial x_i} \frac{\partial L}{\partial a_j}$$

$$= \sum_{j \in Post(i)} w_{ij} g_j$$

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

$$= f'(a_i) \sum_{j \in Post(i)} w_{ij} g_j$$



Training the network: forward and backward propagation

- Special case for Rectified Linear Unit (ReLU) activations

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

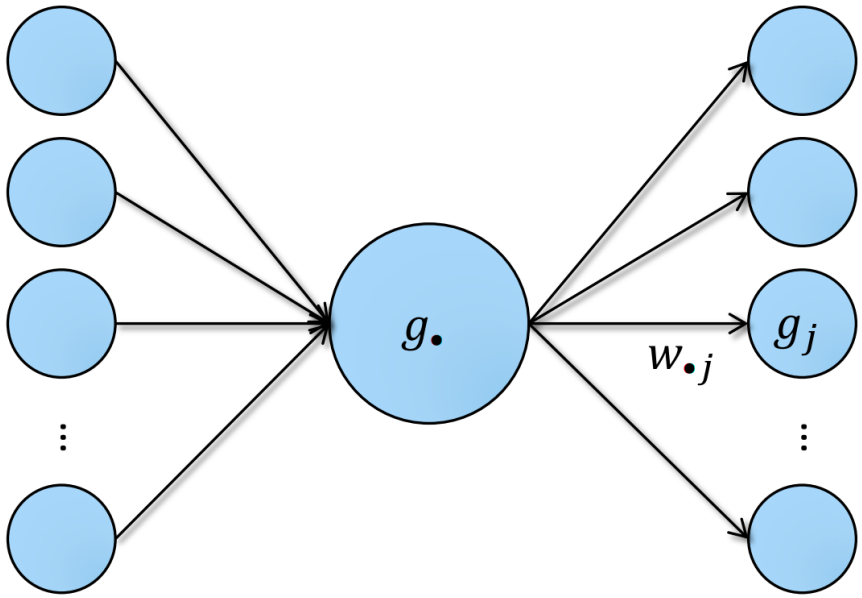
- Sub-gradient is step function

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

- Sum gradients from downstream nodes

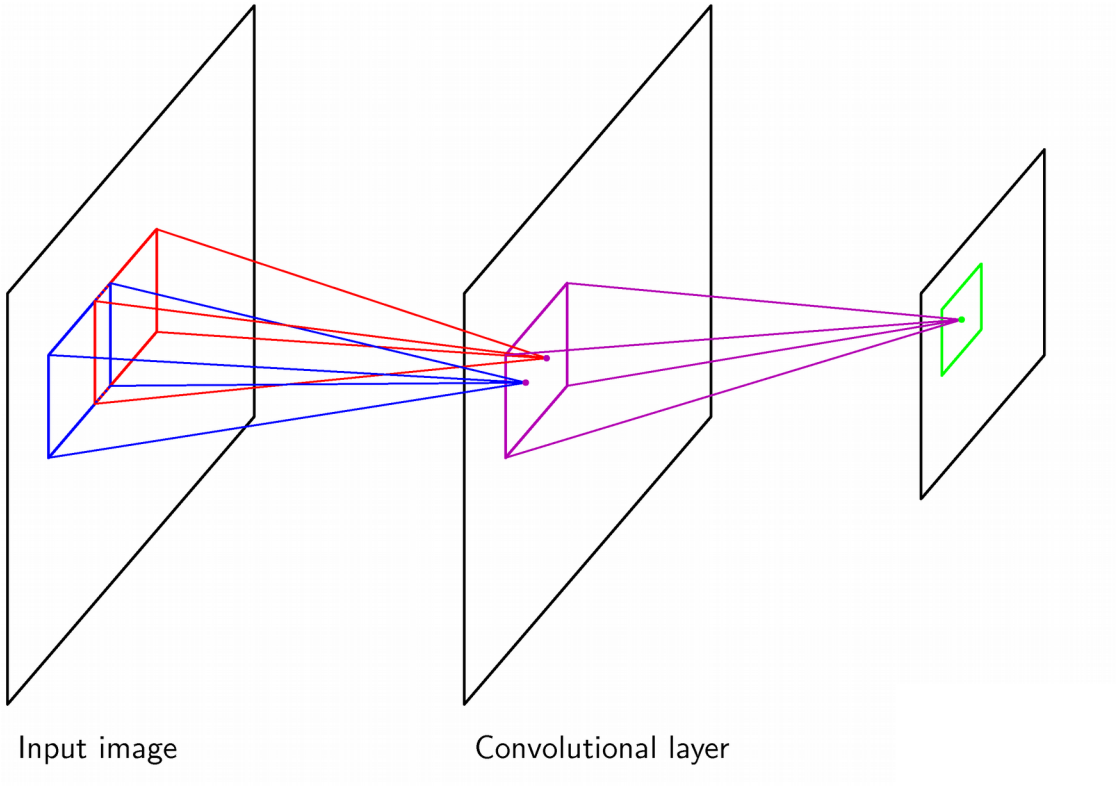
$$g_i = \begin{cases} 0 & \text{if } a_i \leq 0 \\ \sum_{j \in Post(i)} w_{ij} g_j & \text{otherwise} \end{cases}$$

  - Set to zero if in ReLU zero-regime
  - Compute sum only for active units



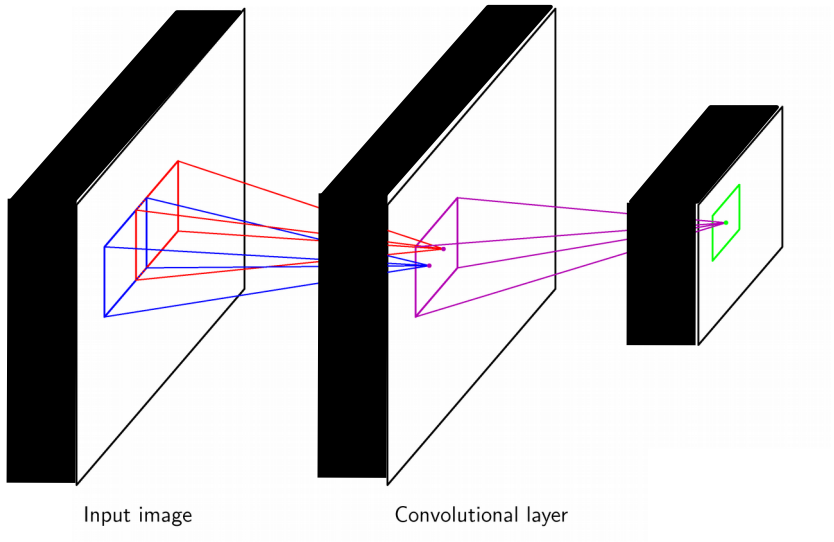
Convolutional neural networks

- 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



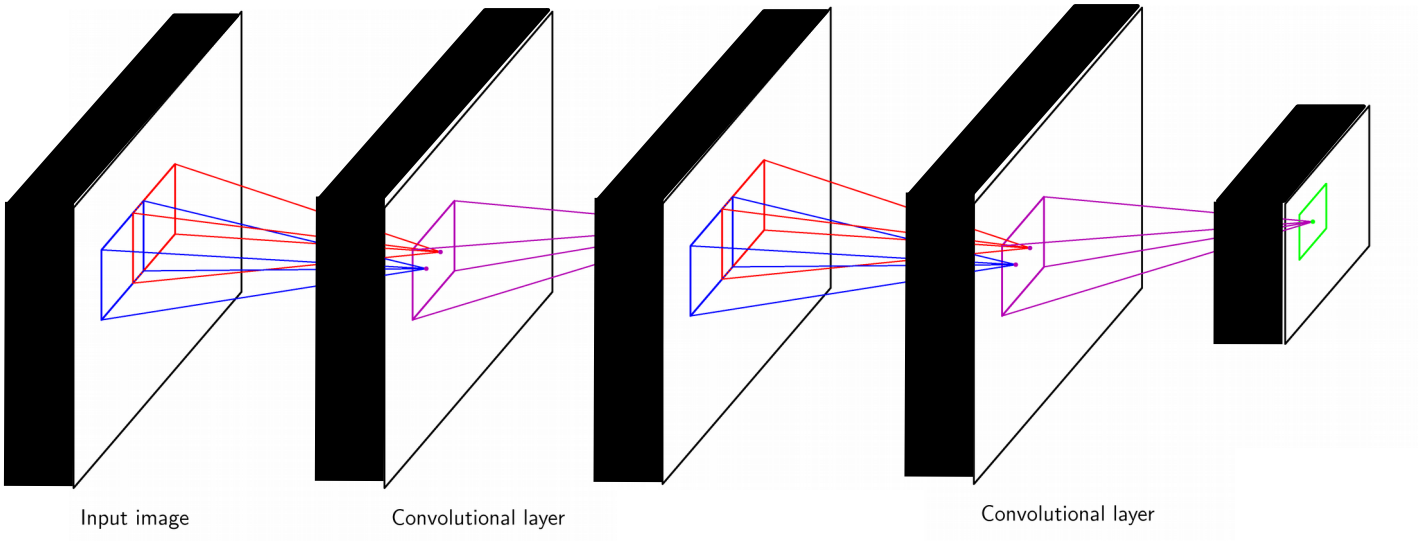
Convolutional neural networks

- Hidden units form another “image” or “response map”
  - Contain result of convolution (linear funcion 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



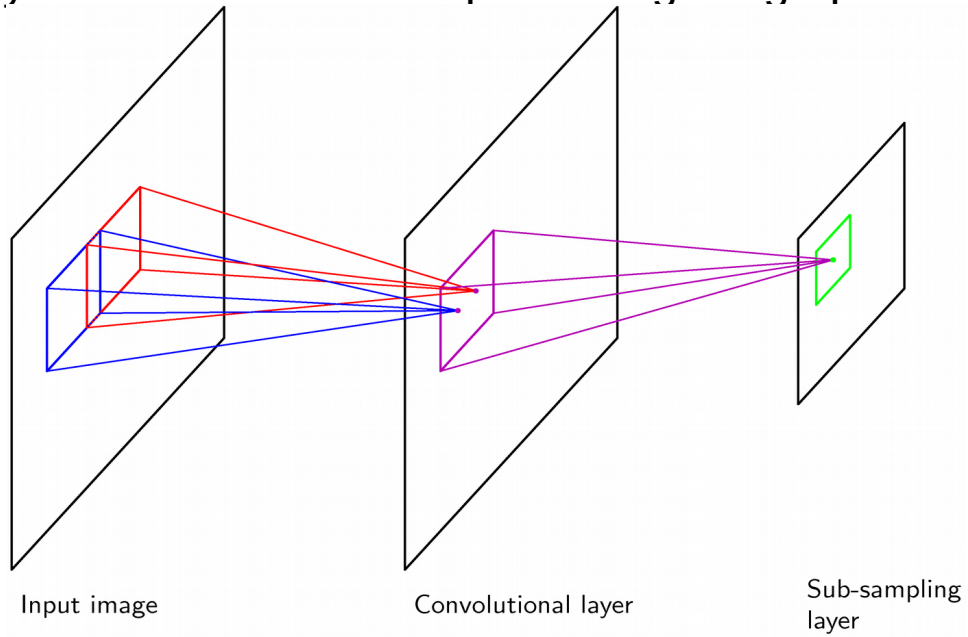
Convolutional neural networks

- “Deep” multi-layer architectures
- “Receptive field” is area in original image impacting a certain unit
  - Receptive field size grows linearly over layers
  - If we use a convolutional filter of size w x w, then each layer the receptive field increases by (w-1)
- Later layers can capture more complex patterns over larger areas



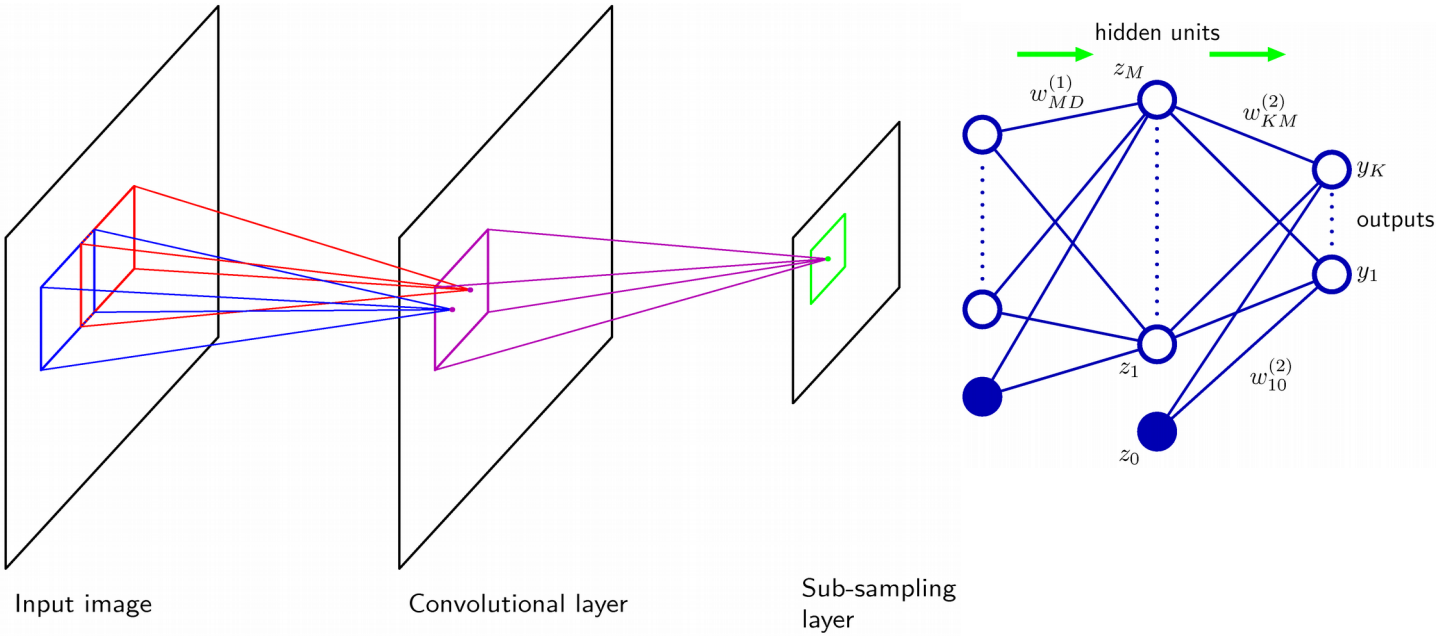
Convolutional neural networks

- Sub-sampling layers reduce the resolution of output map (typically factor 2)
- Pool responses from previous output
  - Max pooling: retain max over receptive field
  - (Weighted) Average pooling
  - Simply drop some pixels, i.e. compute last convolution with certain “stride”
- Receptive field size grows exponentially in number of subsampling layers
  - Only few layers are needed to capture long-range patterns



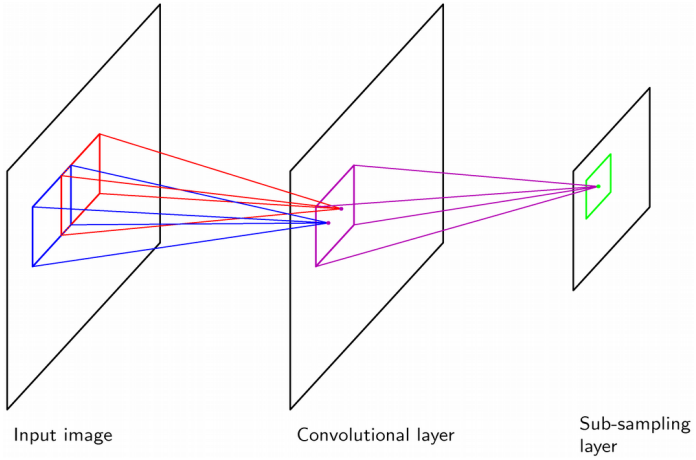
Convolutional neural networks

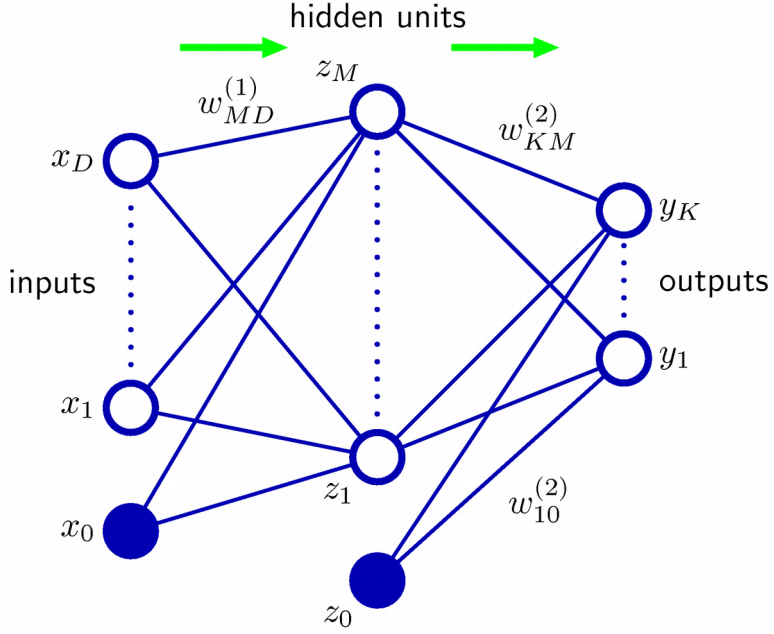
- Convolutional and subsampling layers typically followed by several “fully connected” layers, i.e. standard multi-layer network
- Assembles all local information into global representation that is mapped to response variables
- Softmax over outputs to generate distribution over image class label



Relation to “fully connected” neural networks

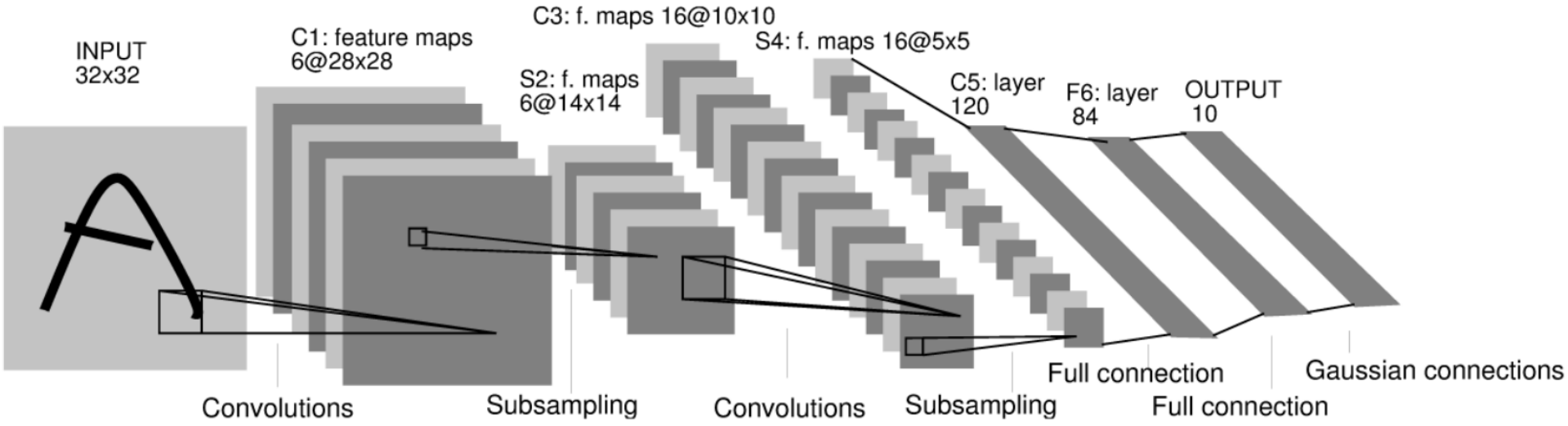
- Hidden units
  - Spatially organized: output of convolution filter at certain position
  - Local connectivity: depend only on small fraction of input units
- Connection weights
  - Same filter weights used for a complete output map
  - Massive weight sharing: nr. of parameters does not grow in output size





Convolutional neural network architectures

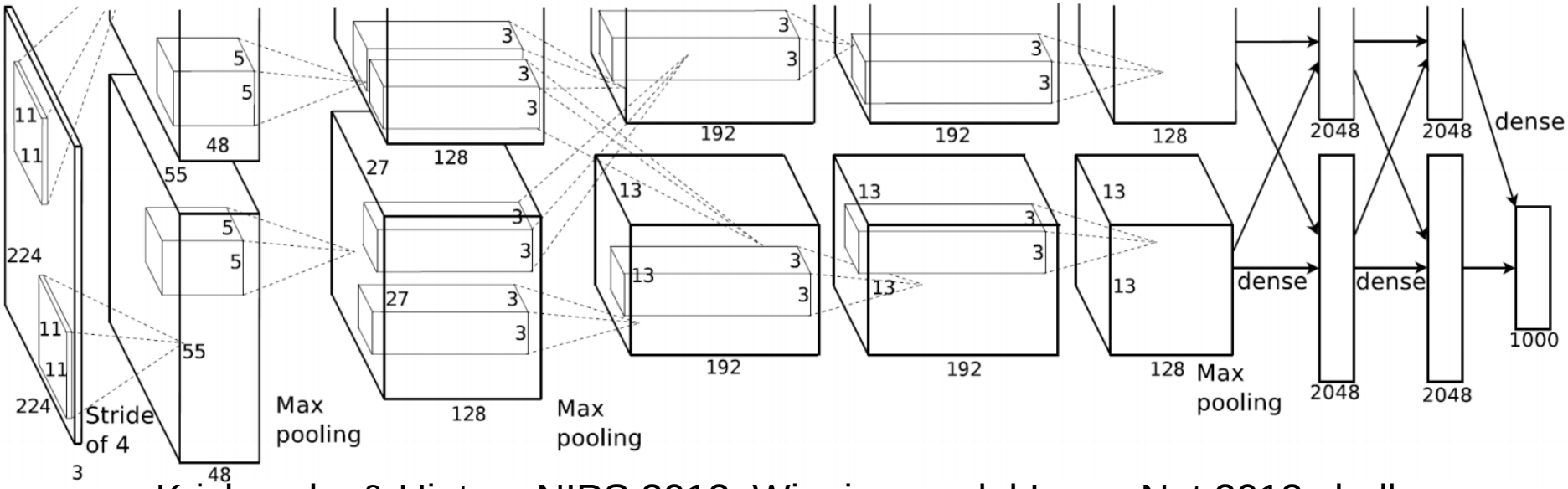
- Convolutional layers: local features along scale and abstraction hierarchy
  - Convolutions: number of filters, and filter sizes
  - Nonlinearity
  - Subsampling: scale factor, and pooling function
- Fully connected layers: assemble local features into global interpretation
  - Multi-layer perceptron



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

Convolutional neural network architectures

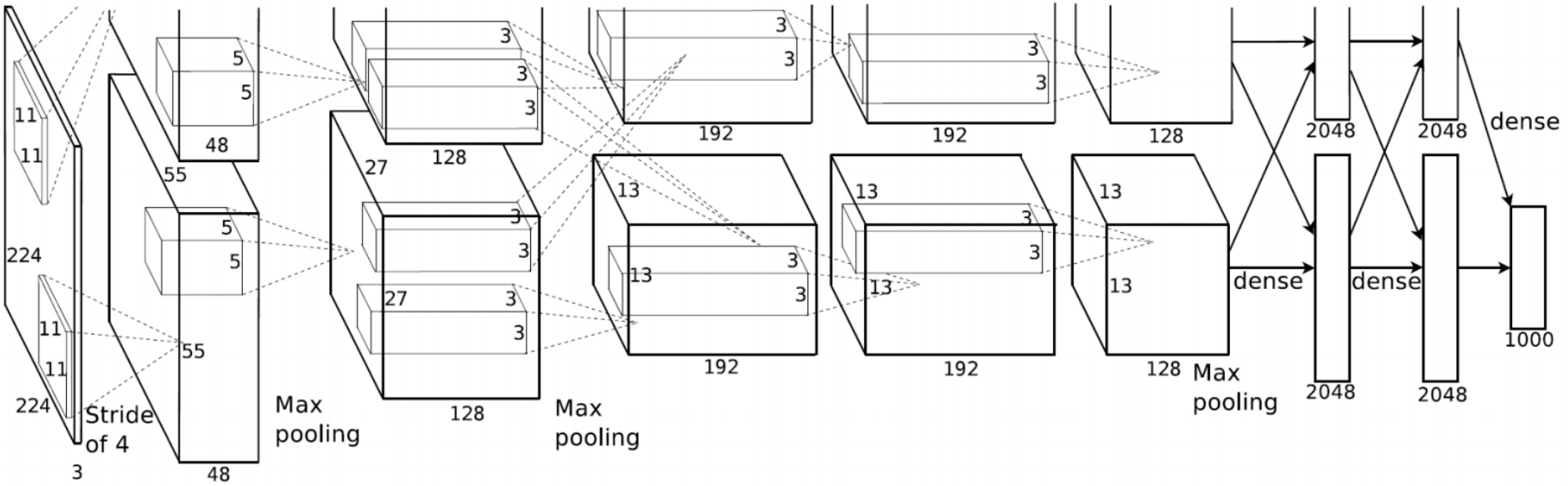
- Similar architectures as digit recognition network
  - ReLU activations instead of hyperbolic tangent
  - Deeper: e.g. 19 layers in Simonyan & Zisserman, ICLR 2015
  - Wider: More filters per layer: hundreds instead of tens
  - Wider: thousands of nodes in fully connect layers
  - Number of parameters: around 60 million (!)



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

Convolutional neural network architectures

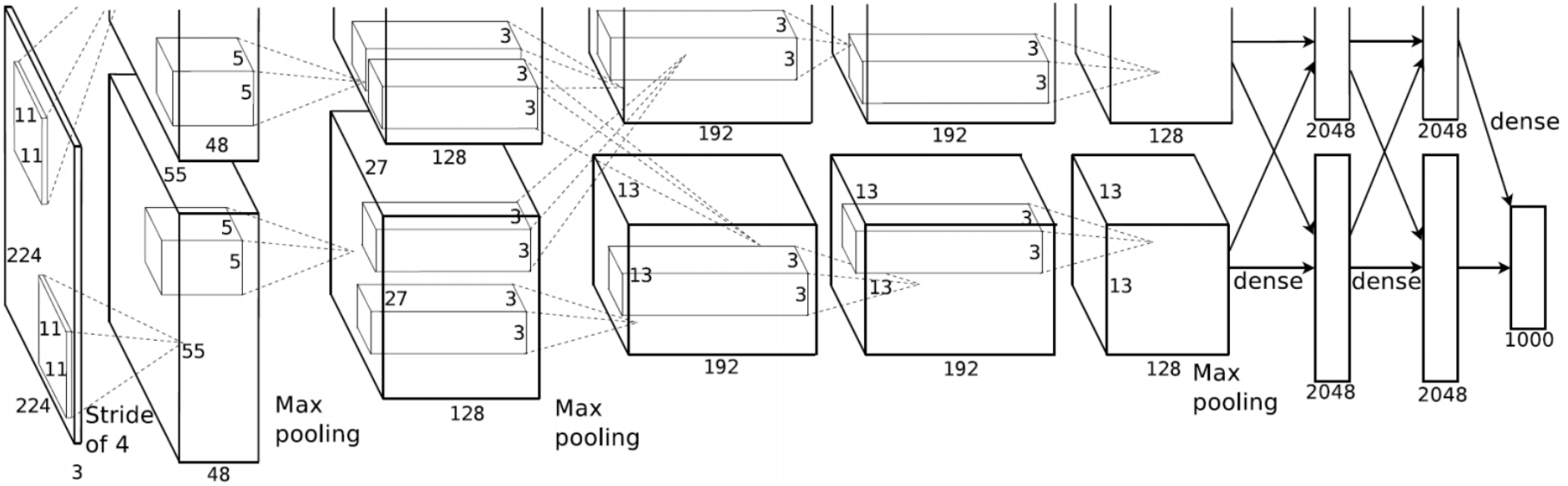
- Similar architectures for general object recognition a decade later
- More training data
  - 1.2 millions of 1000 classes for ImageNet challenge
  - 200 million faces in Schroff et al, CVPR 2015
- GPU-based implementations
  - Massively parallel computation of convolutions
  - Krizhevsky & Hinton, 2012: six days of training on two GPUs



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

Understanding convolutional neural network activations

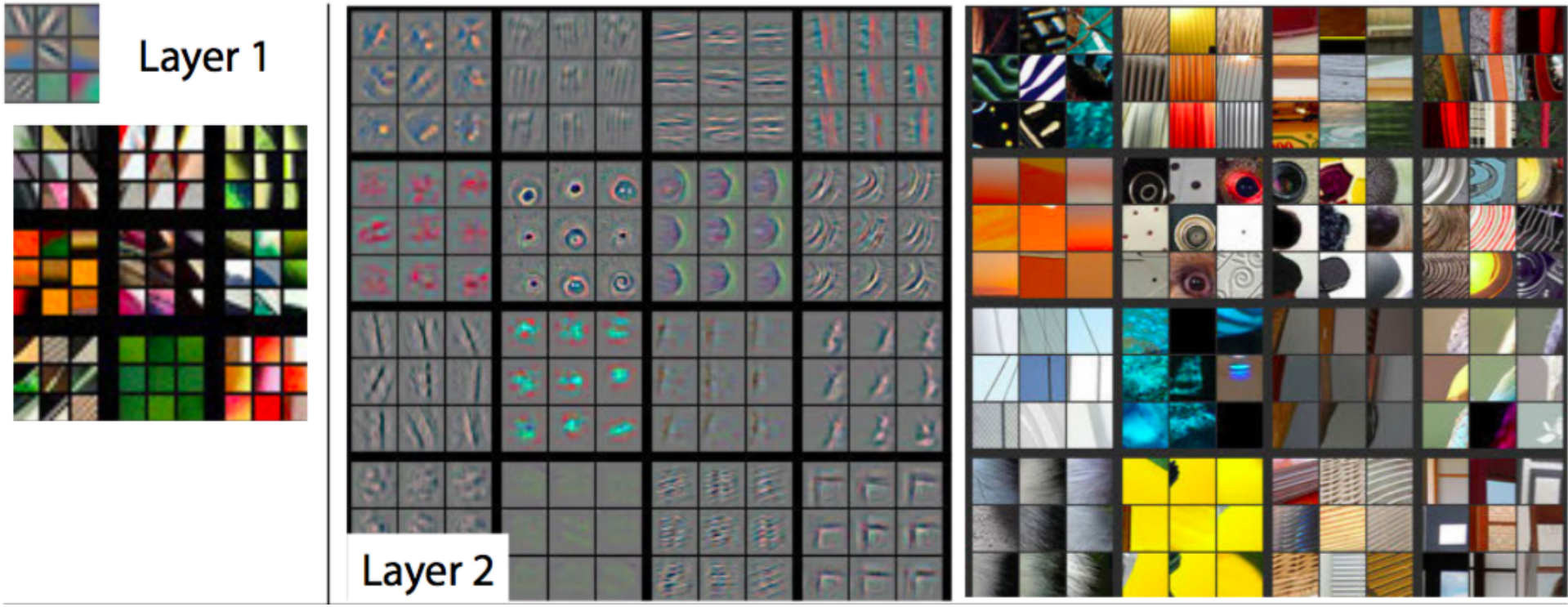
- Architecture consists of
  - 5 convolutional layers
  - 2 fully connected layers
- Visualization of patches that yield maximum response for certain units
  - We will look at each of the 5 convolutional layers



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

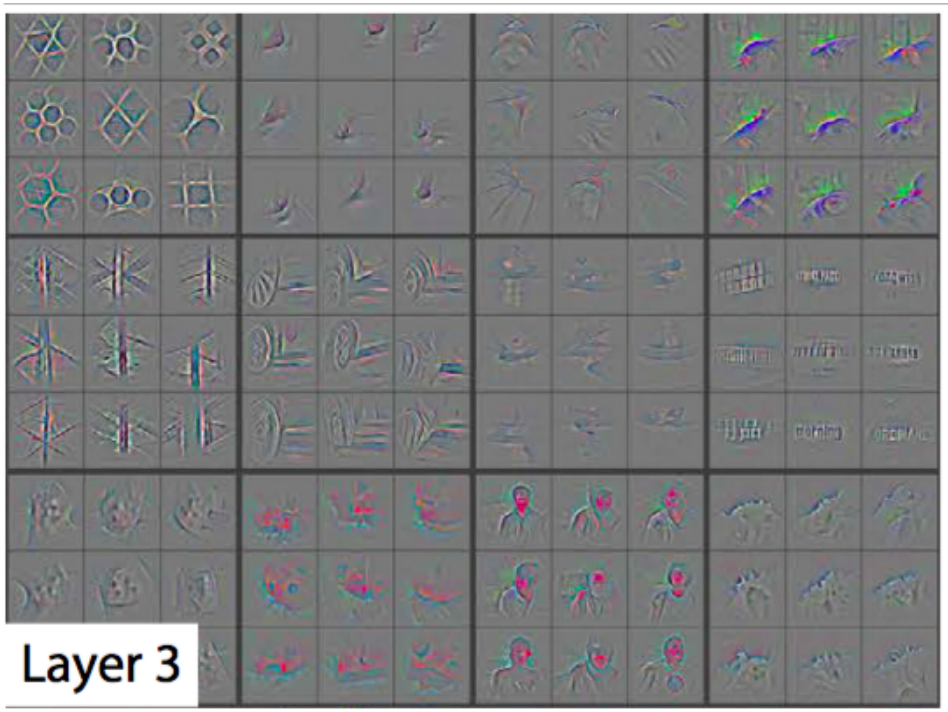
Understanding convolutional neural network activations

- Patches generating highest response for a selection of convolutional filters, and reconstruction of these patches based on network weights
  - Zeiler and Fergus, ECCV 2014
- Layer 1: simple edges and color detectors
- Layer 2: corners, center-surround, ...



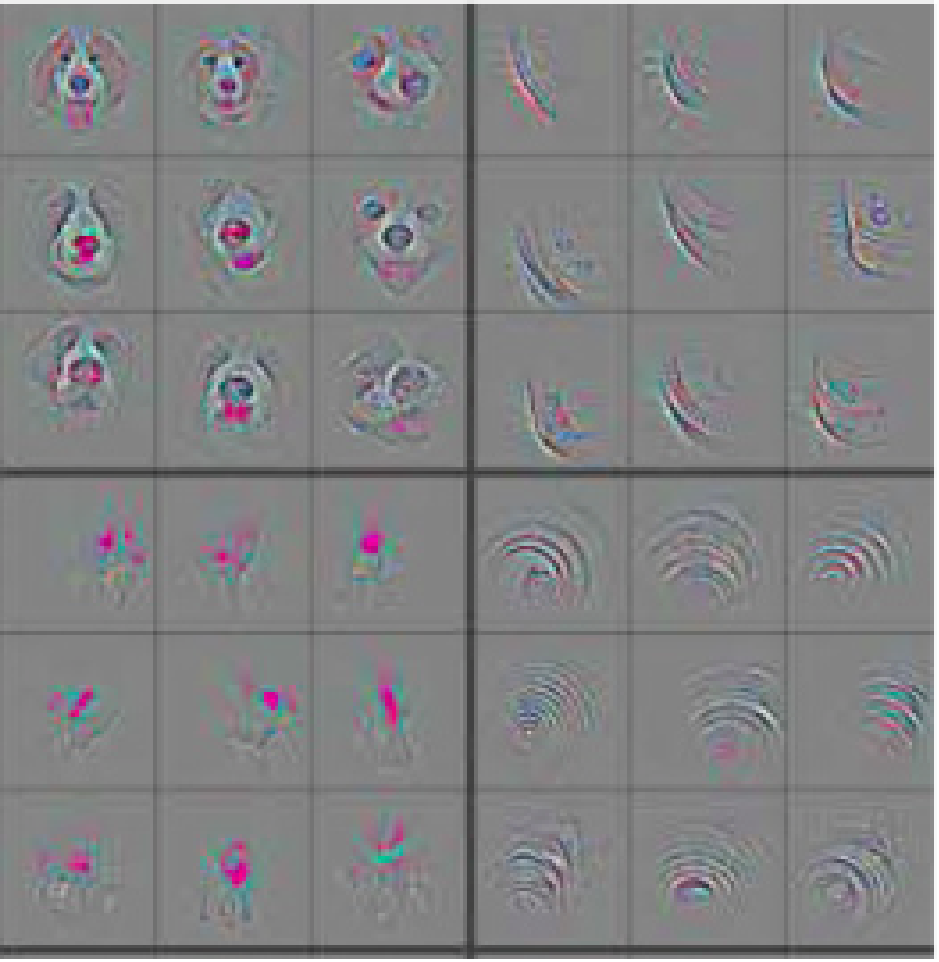
Understanding convolutional neural network activations

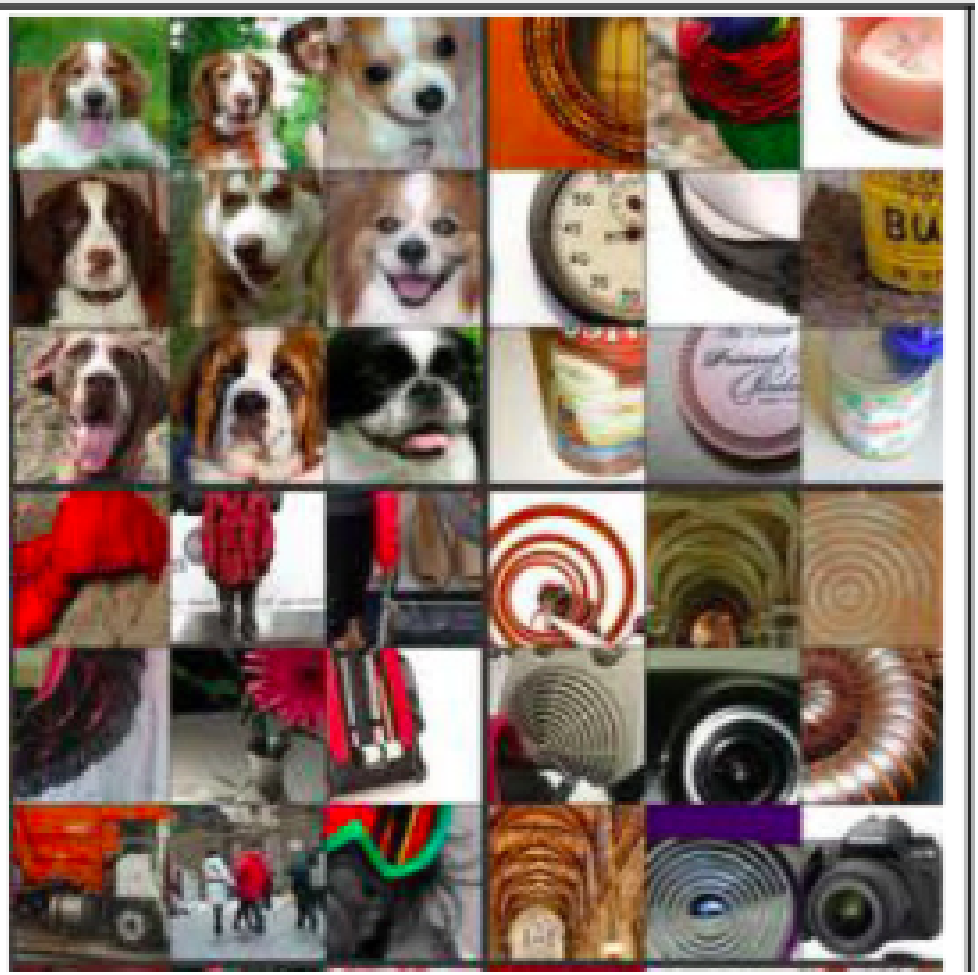
- Layer 3: textures, object parts



Understanding convolutional neural network activations

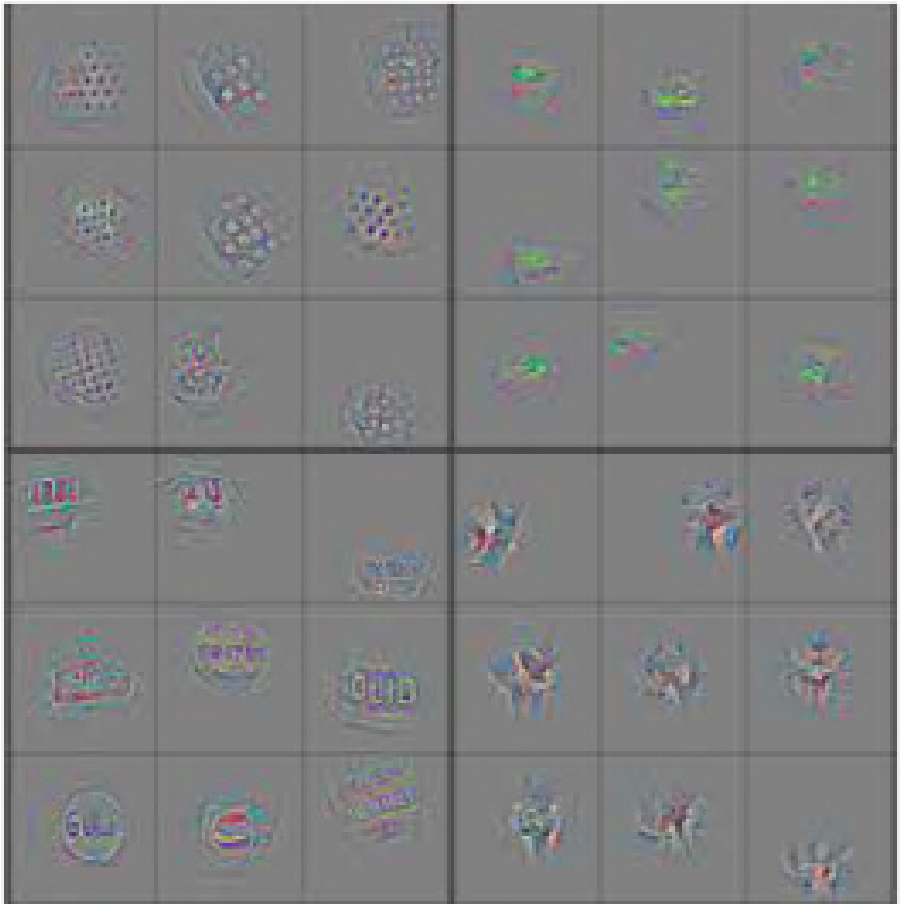
- Layer 4: complex textures and object parts
  - Invariance for backgrouns: suppression in reconstruction

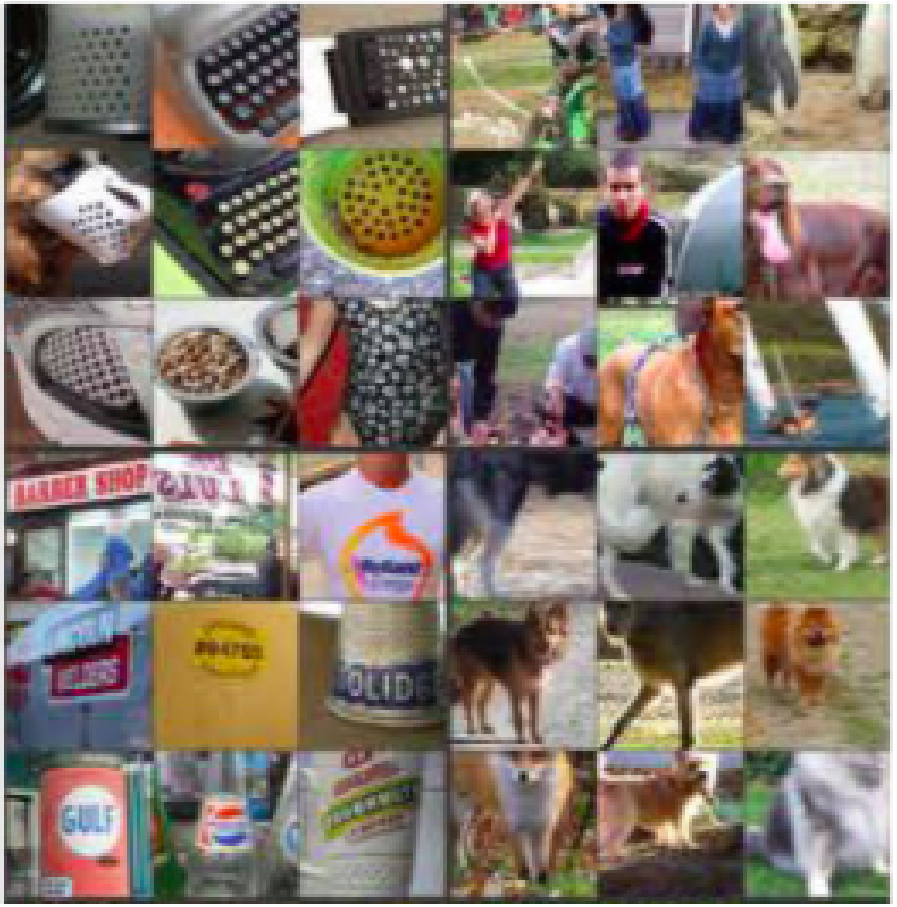




Understanding convolutional neural network activations

- Layer 5: complex textures and object parts



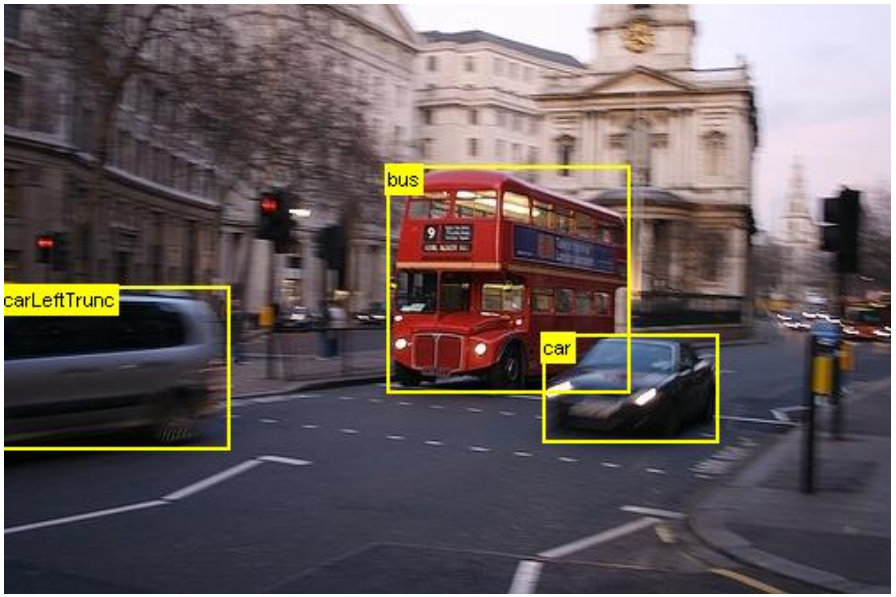


Understanding convolutional neural network activations

- Regions with max response for Conv-5 filters with positive weights for cats, negative weights for cats, and positive for sheep and person
  - Girshick et al., CVPR 2014

Convolutional neural networks for other tasks

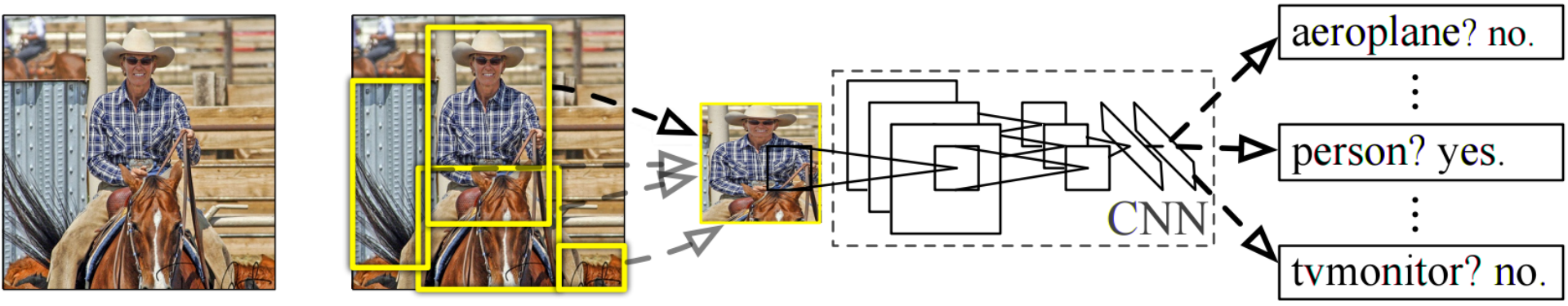
- **Object category localization**



- Semantic segmentation

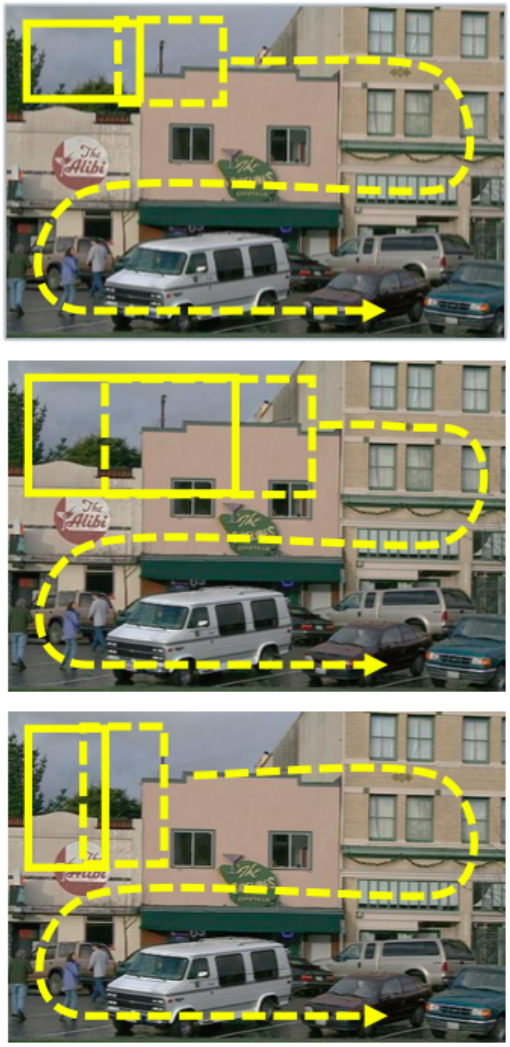
CNNs for object category localization

- Apply CNN image classification model to image sub-windows
  - For each window decide if it represents yes/no a “car”, “sheep”, ...
- Map detection windows to 224 x 224 image to fit CNN model
- Unreasonably many image regions to consider if applied in naive manner

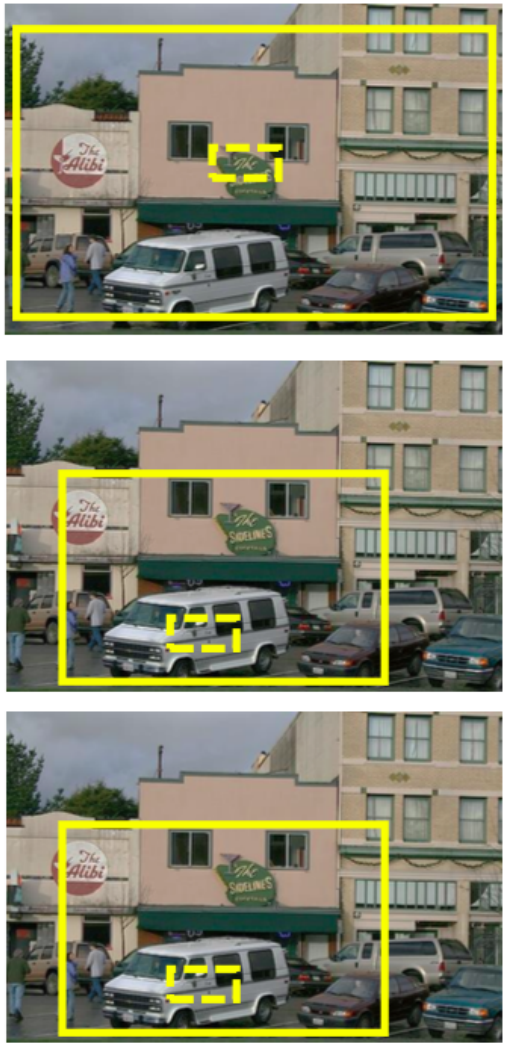


R-CNN, Girshick et al., CVPR 2014

How to avoid exhaustive sliding window search



Sliding window
(Viola and Jones 2002;
Felzenszwalb *et al*. 2008, ...)

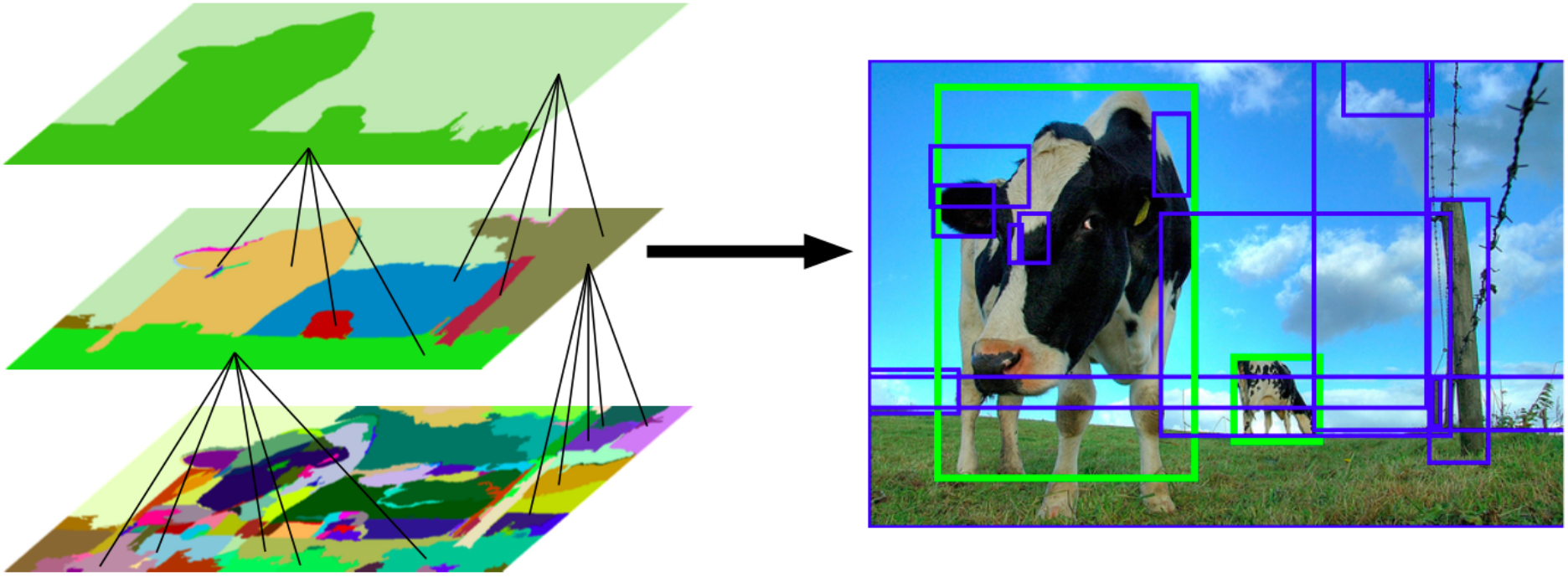


Branch & bound
(Lampert *et al.* 2008;
Lehmann *et al.* 2013)

Selective Search
(Alexe *et al.* 2010;
Sande *et al.* 2011)

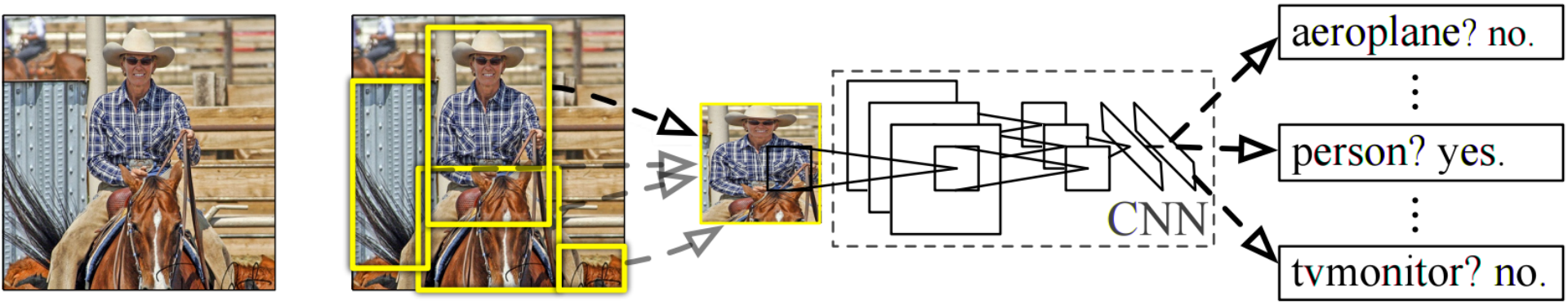
Search: restricted scanning of bounding box space

- 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

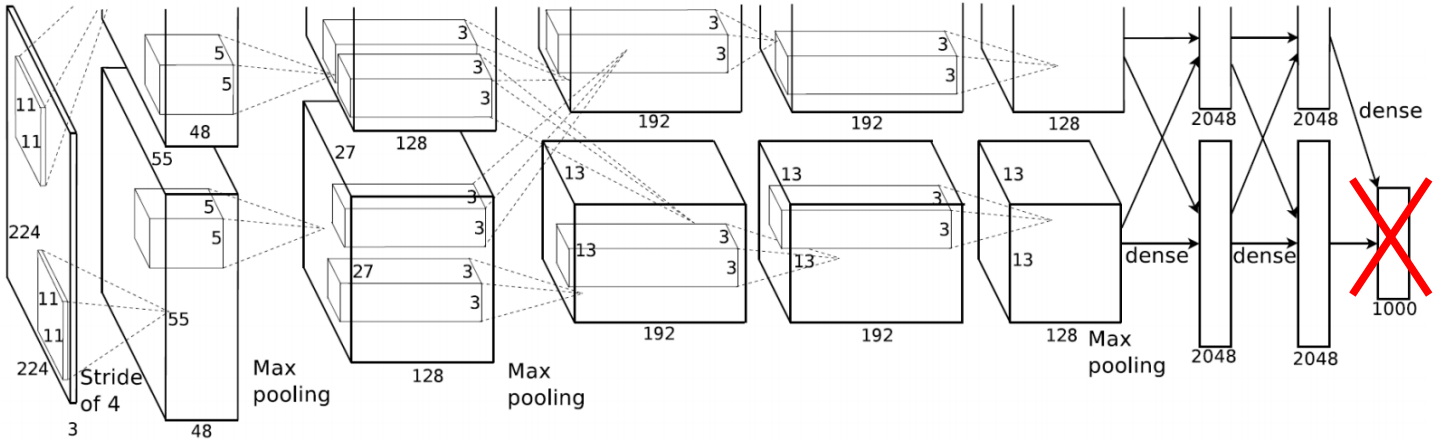
- Apply CNN image classification model to image sub-windows
  - For each window decide if it represents yes/no a “car”, “sheep”, ...
- Unreasonably many image regions to consider if applied in naive manner
  - Use detection proposal mechanism
  - Based on low-level image contours propose a (few) thousand windows

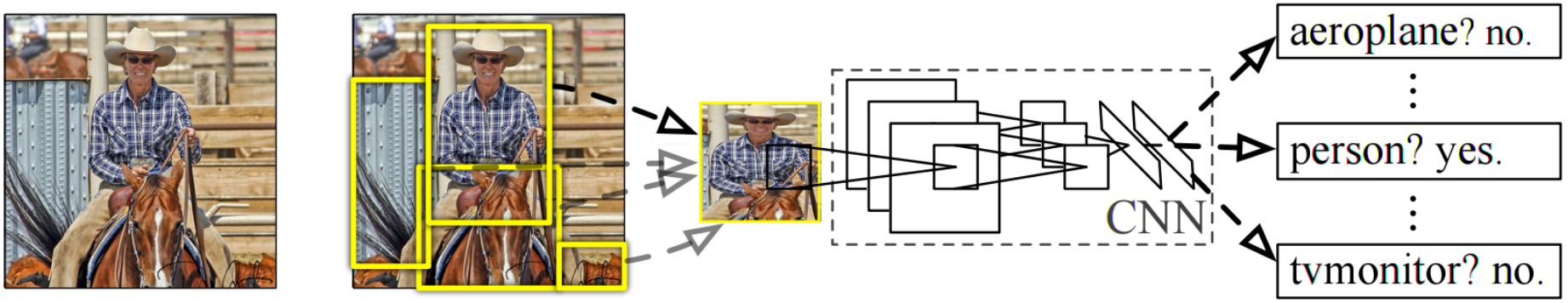


R-CNN, Girshick et al., CVPR 2014

CNNs for object category localization

- Too little training data to learn CNN from scratch
  - Only few hundred objects instances labeled with bounding box
  - Pre-train on ImageNet classification problem: Krizhevsky model
  - Replace last classification layer with classification over N categories + background
  - Update CNN weights for classification of detection proposals





CNNs for object category localization

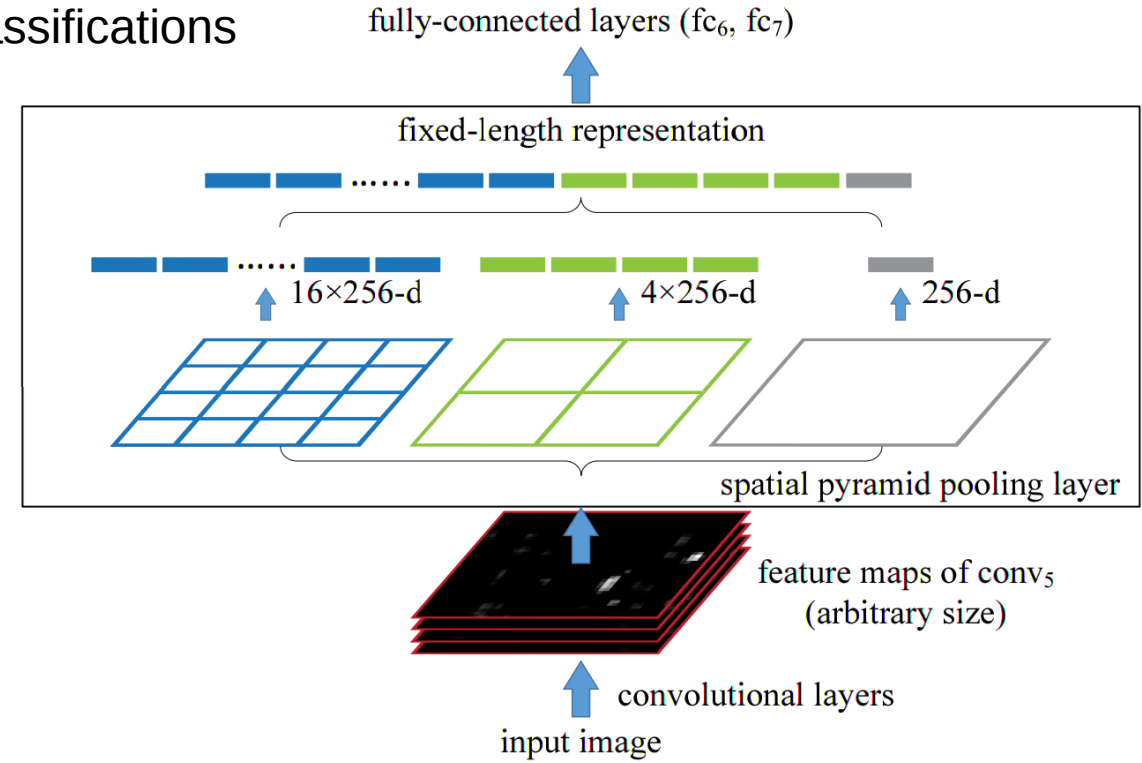
- Comparison with state of the art non-CNN models
  - Object detection is correct if window has intersection/union with ground-truth window of at least 50%
- Significant increase in performance of 10 points mean-average-precision (mAP)

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 across image once
- For each window:
  - Pool convolutional features using max-pooling into fixed-size representation
  - Fully connected layers computed per window
  - Output multiple binary classifications



SPP-net, He et al., ECCV 2014

Efficient object category localization with CNN

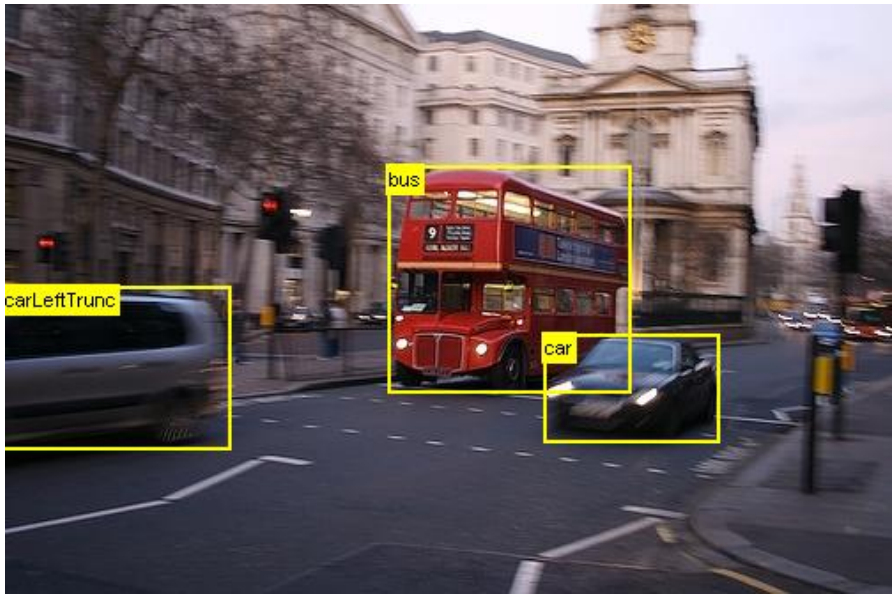
- Refinement: Compute convolutional filters at multiple scales
  - For given window use scale at which window has roughly size 224x224
- Similar performance as explicit window rescaling, and re-computing convolutional filters
- Speedup of about 2 orders of magnitude

	SPP (1-sc) (ZF-5)	SPP (5-sc) (ZF-5)	R-CNN (ZF-5)
ftfc_7	54.5	55.2	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 (*vs.* 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

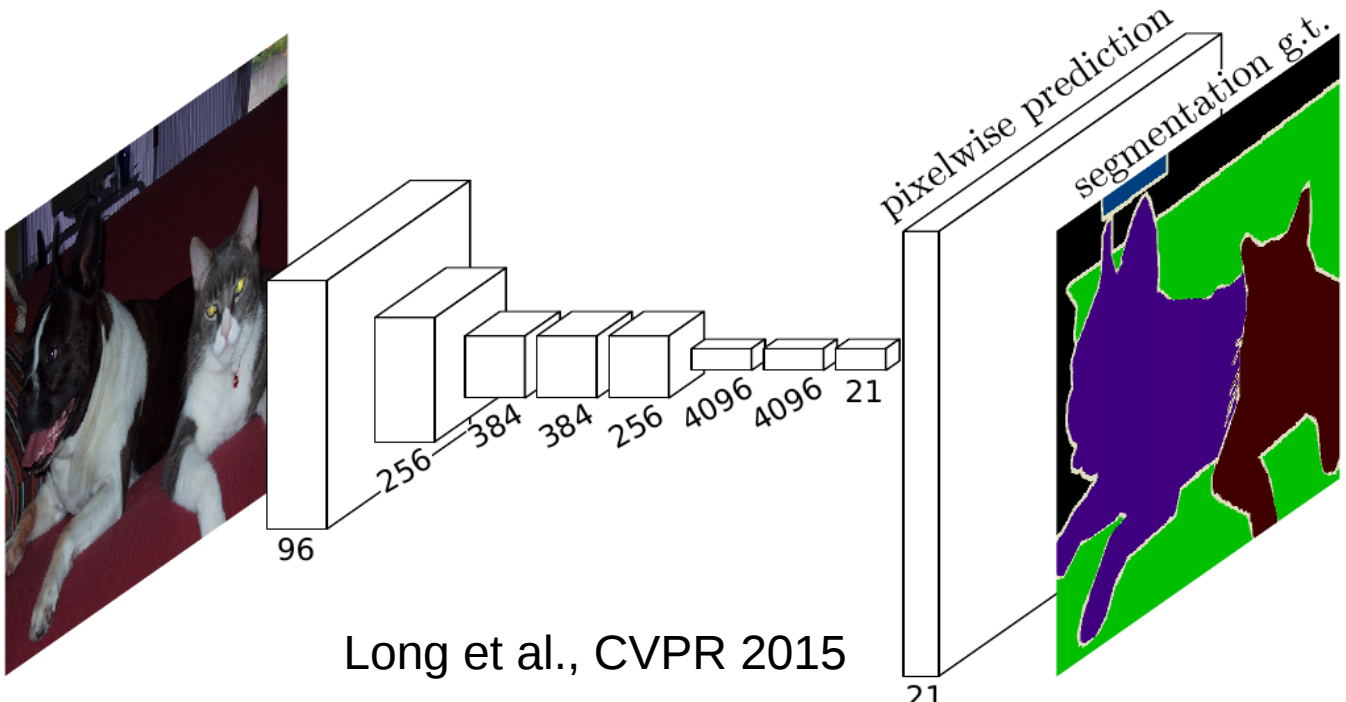
- Object category localization



- **Semantic segmentation**

Application to semantic segmentation

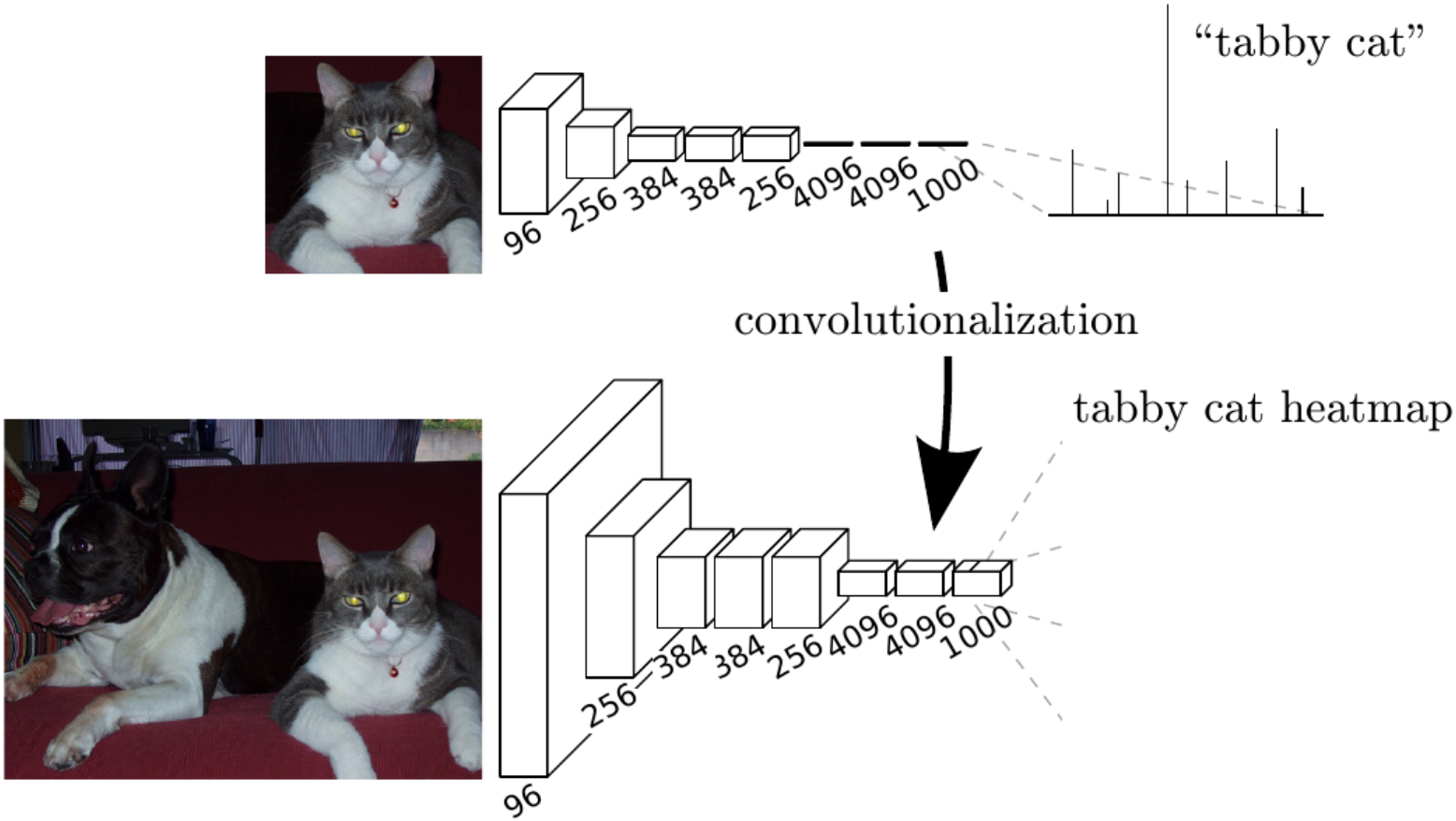
- Assign each pixel to an object or background category
  - Consider running CNN on small image patch to determine its category
  - Train by optimizing per-pixel classification loss
- Similar to SPP-net: avoid wasteful re-computation of convolutional filters
  - Compute convolutional layers once per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



Long et al., CVPR 2015

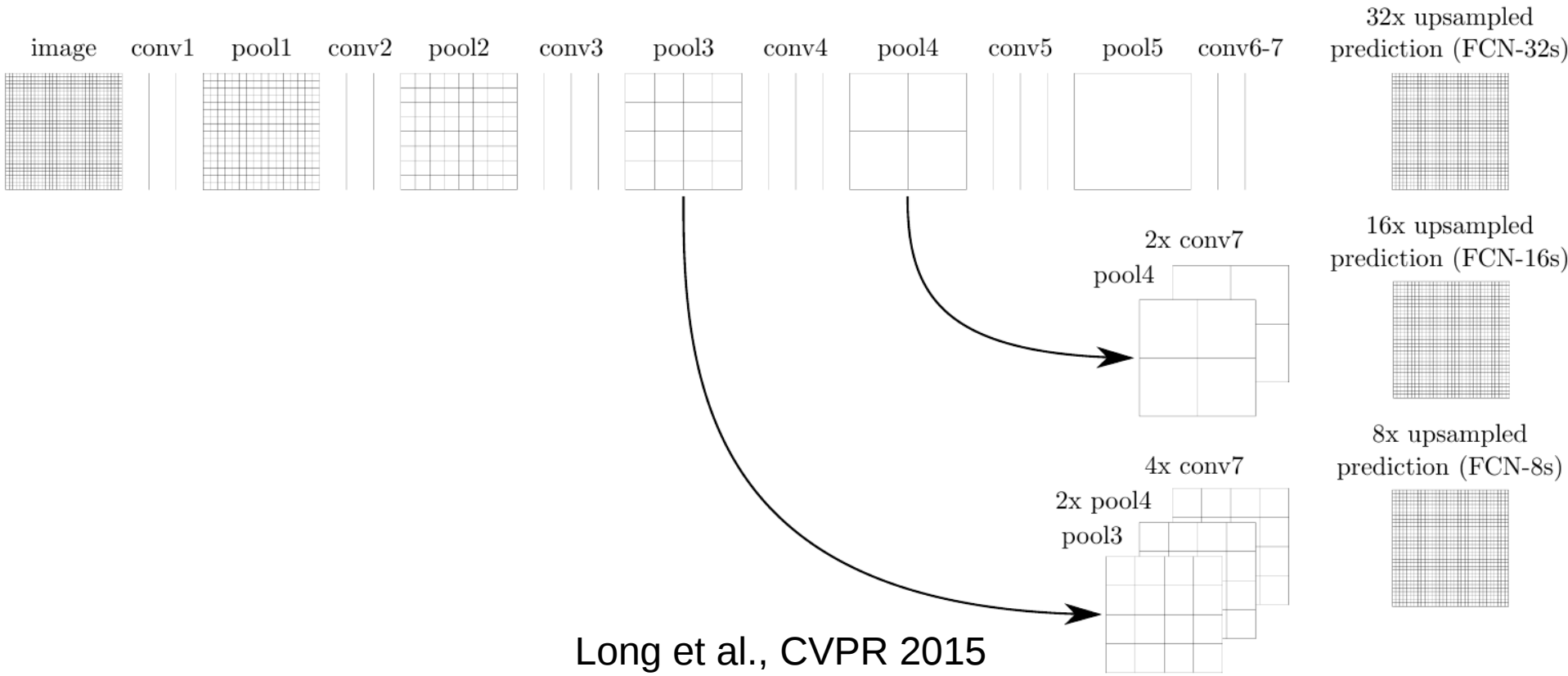
Application to semantic segmentation

- 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



Application to semantic segmentation

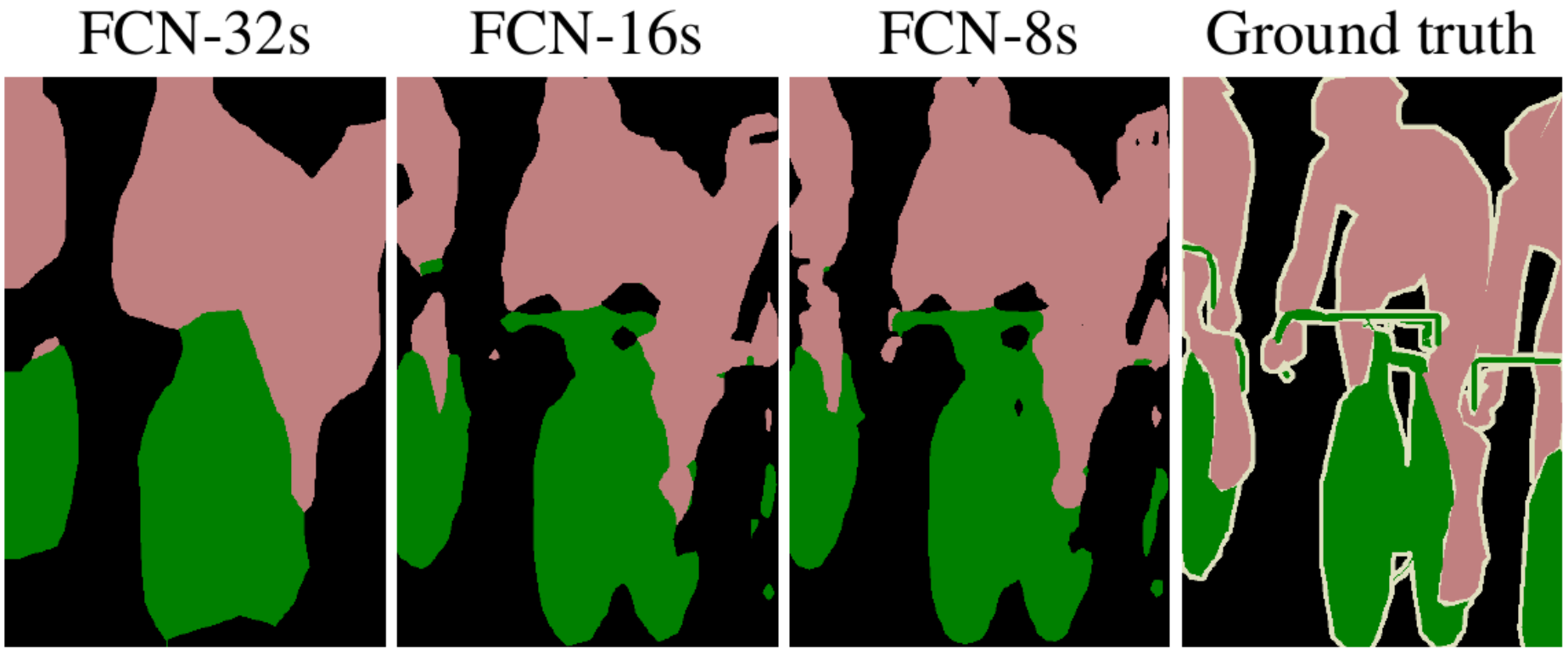
- Idea 1: up-sampling via bi-linear interpolation, gives blurry predictions
- Idea 2: weighted sum of response maps at different resolutions
  - Up-sampling of coarser level with (with learned upsampling) mask
  - Train all layers in integrated manner
- Train first at coarse level to initialize network with finer resolutions



Long et al., CVPR 2015

Application to semantic segmentation

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



Application to semantic segmentation

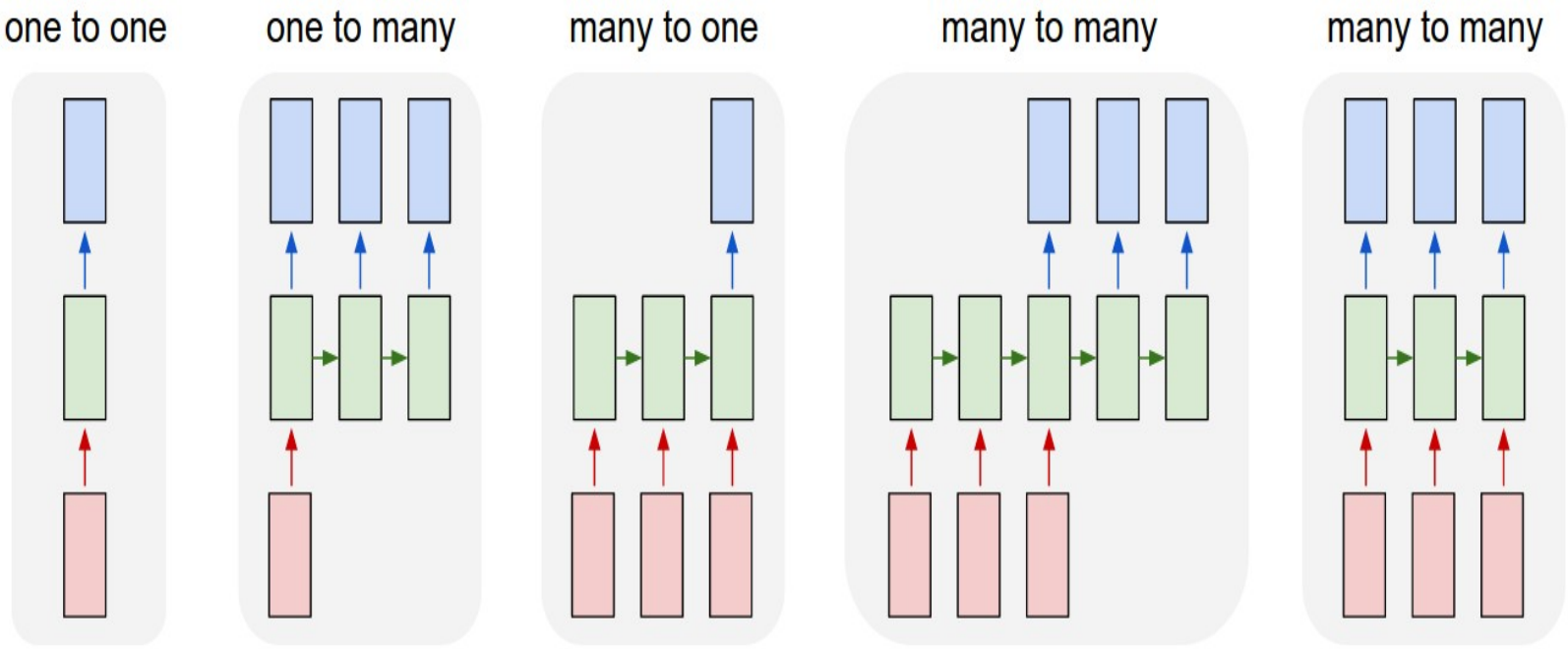
- Further improvements possible using conditional random field (CRF) models
- Beyond independent prediction of individual pixel labels
- Zheng et al., ICCV 2015

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
- Applications for object localization and semantic segmentation
  - Local classification at level of detection windows or pixels
  - Computation of low-level convolutions can be shared across regions

Modeling sequential data

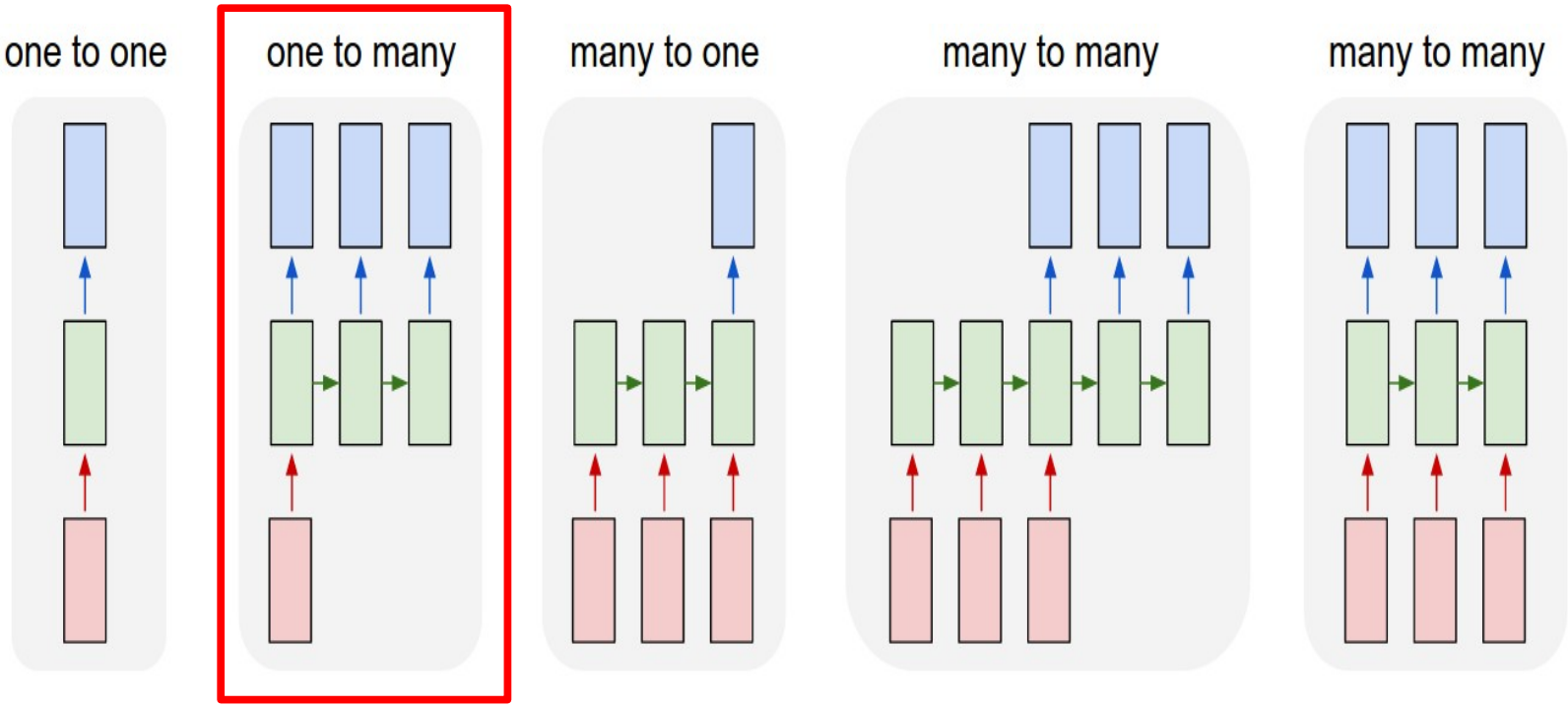
- So far we mostly considered “one-to-one” prediction tasks
  - Classification: one image to one class label, which digit is displayed 0...9
  - Regression: one image to one scalar, how old is this person?
- Many prediction problems have a sequential nature to them
  - Either in input, in output, or 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



Modeling sequential data

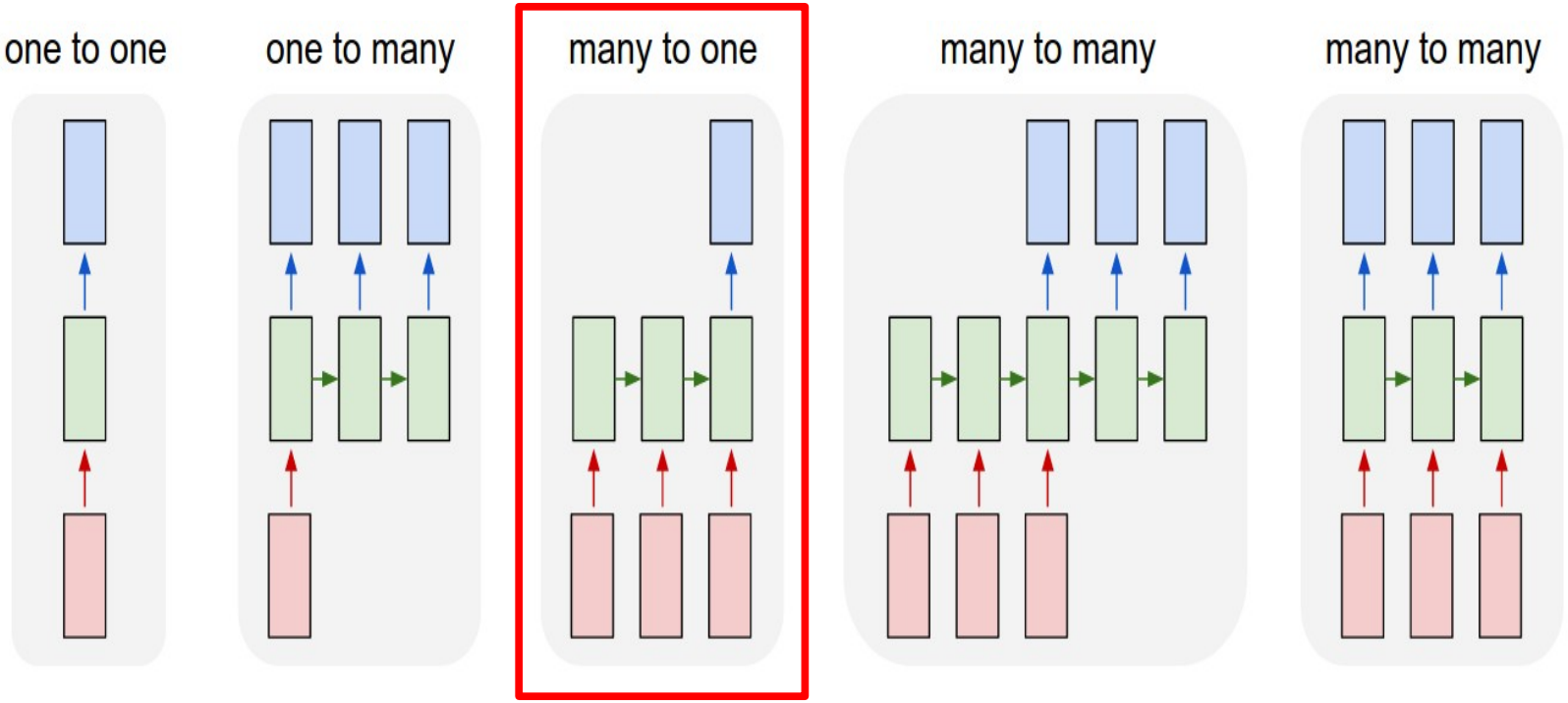
- Image captioning
  - Input: a sentence
  - Output: user rating

I liked it...

Author: JustMeOnline from Portugal

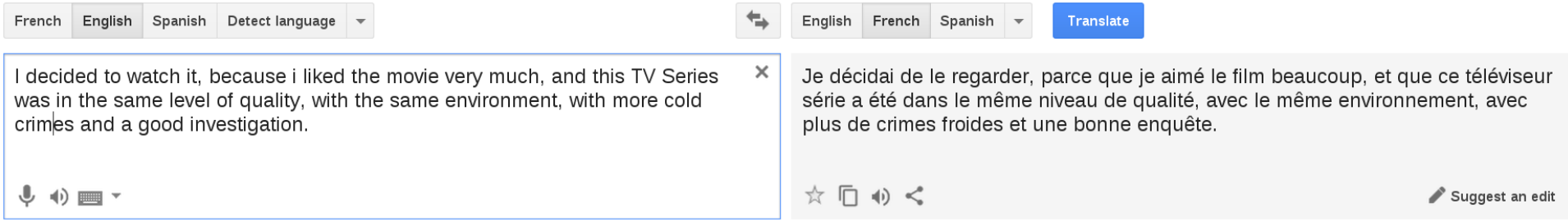
14 November 2014

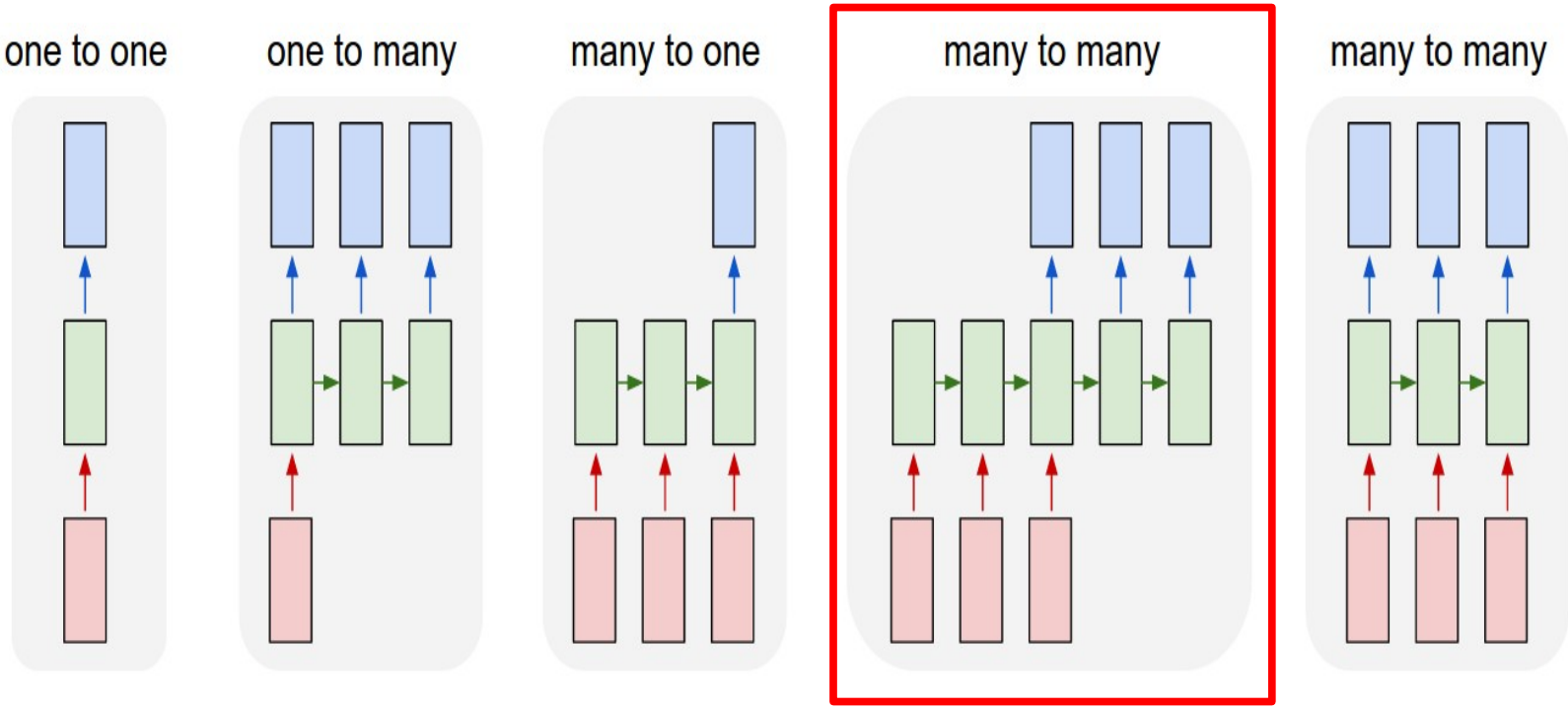
I decided to watch it, because i liked the movie very much, and this TV Series was in the same level of quality, with the same environment, with more cold crimes and a good investigation...



Modeling sequential data

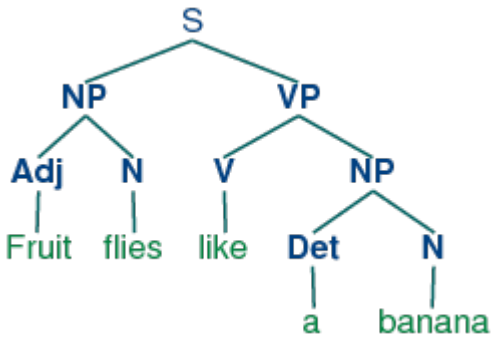
- Translation of a sentence into another language

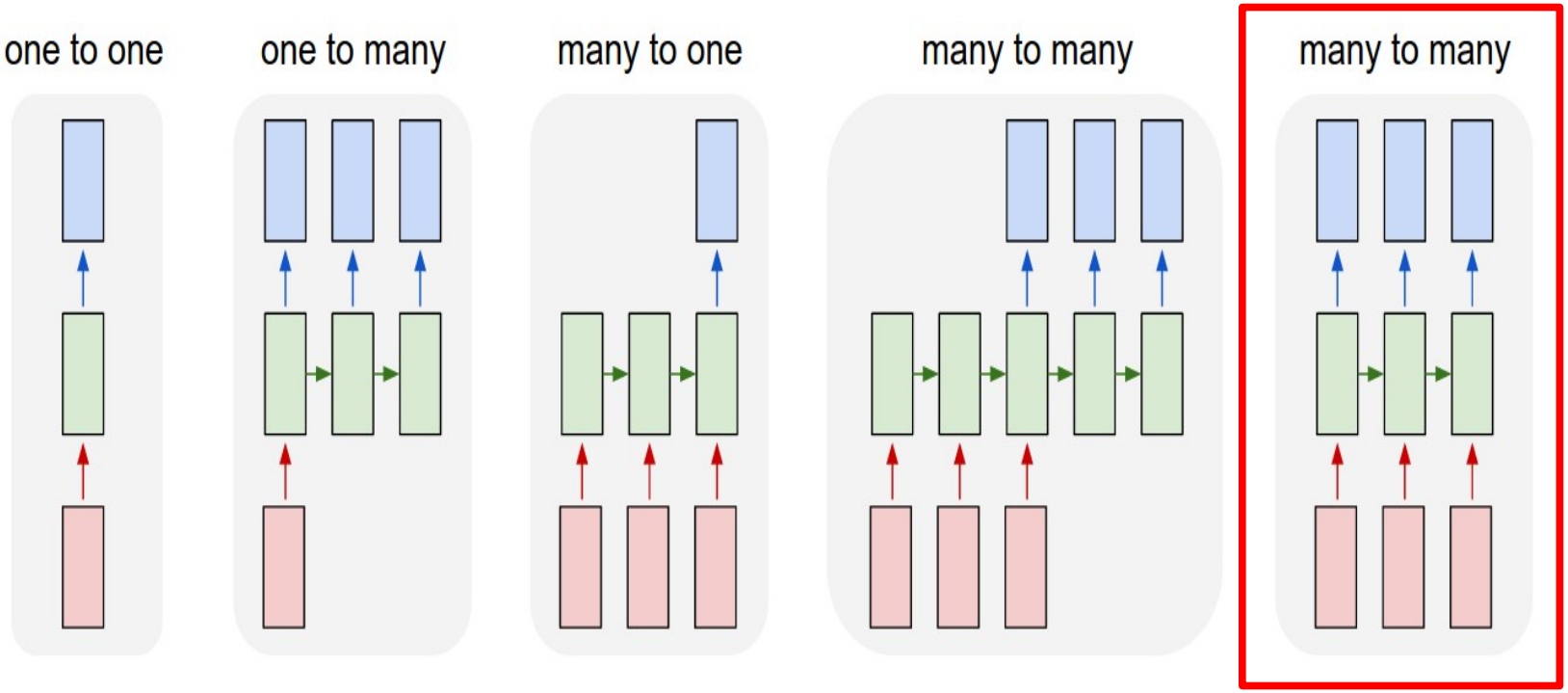




Modeling sequential data

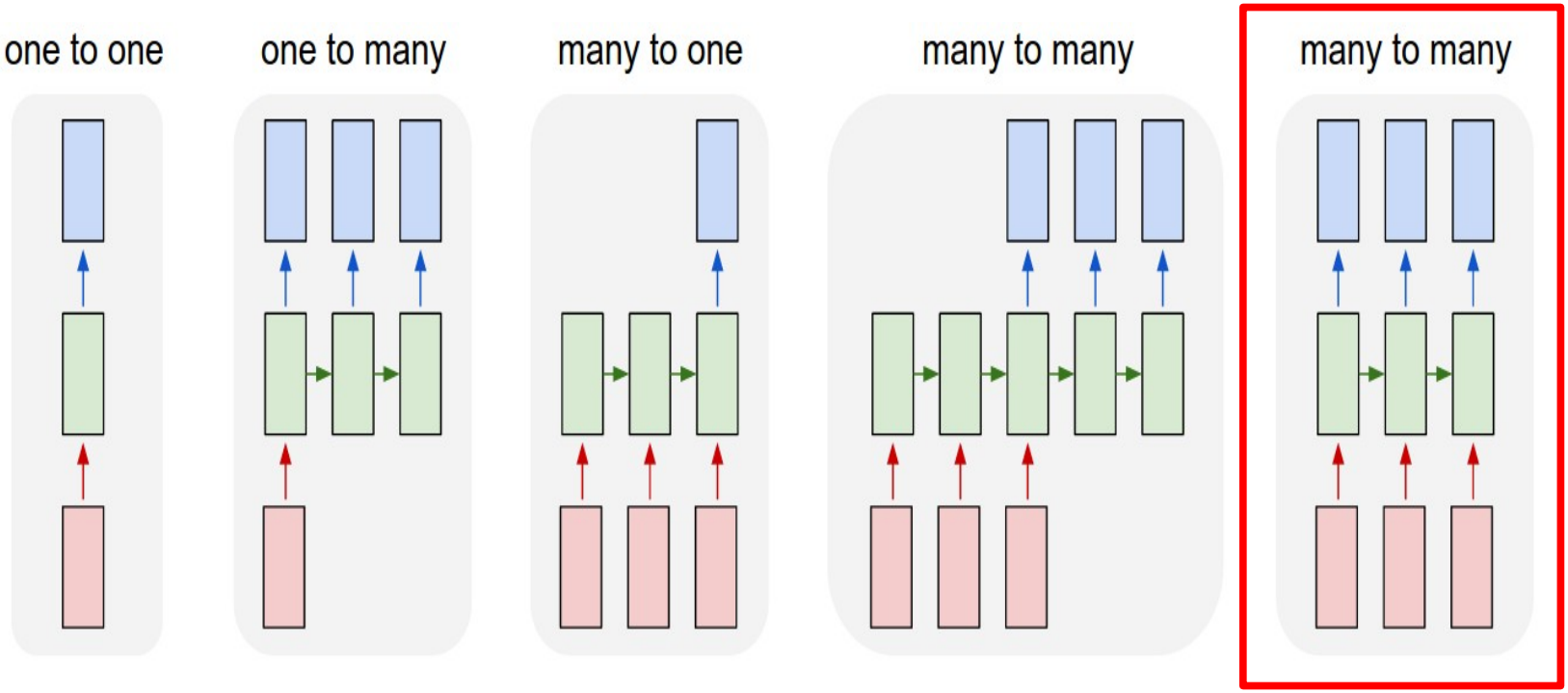
- Part of speech tagging
  - Input: a sentence
  - Output: a part of speech tag for each word





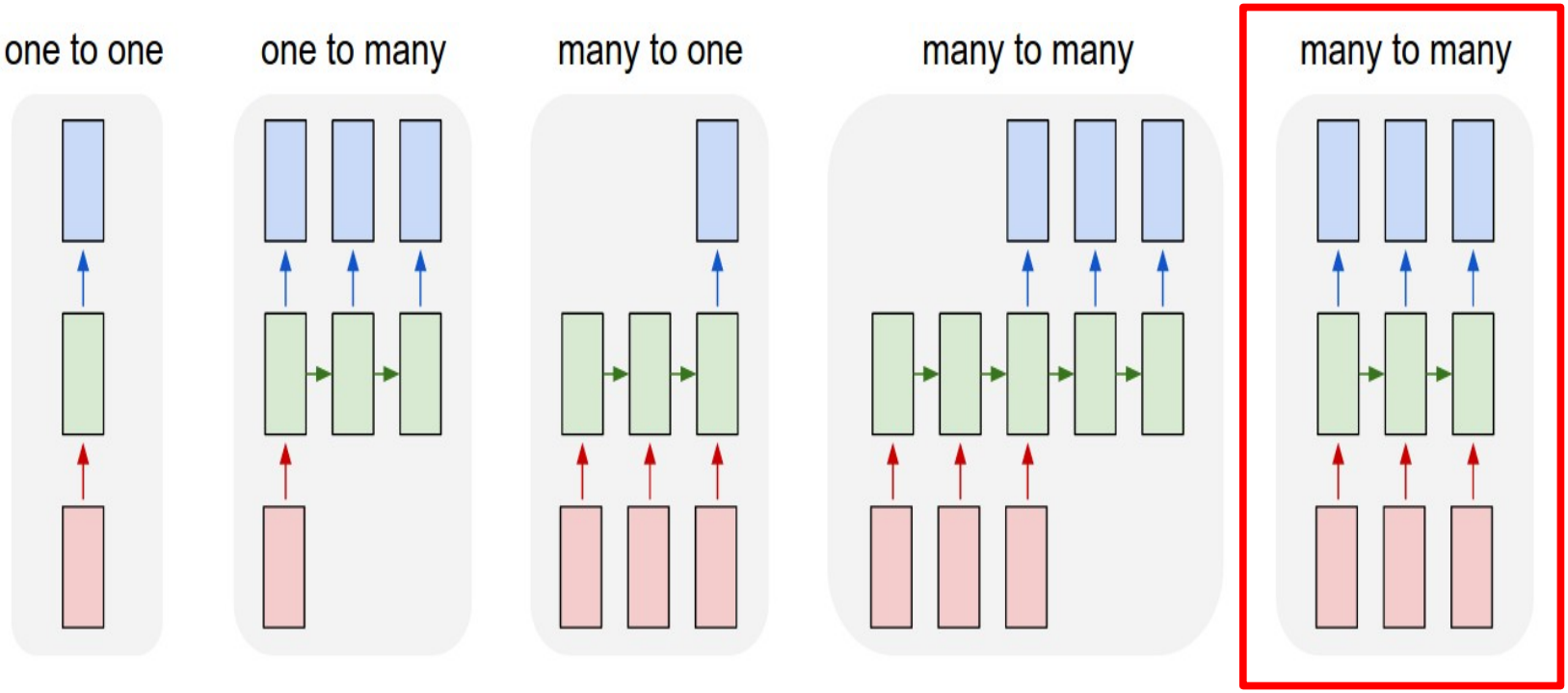
Modeling sequential data

- We could also use an order k Markov chains over either input, output, or both
  - Would result in a “memory” of k time steps only
- The hidden layer allows to build up a representation of the sequence as a whole over time



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 for arbitrarily long sequences
  - Across space in convolutional neural networks

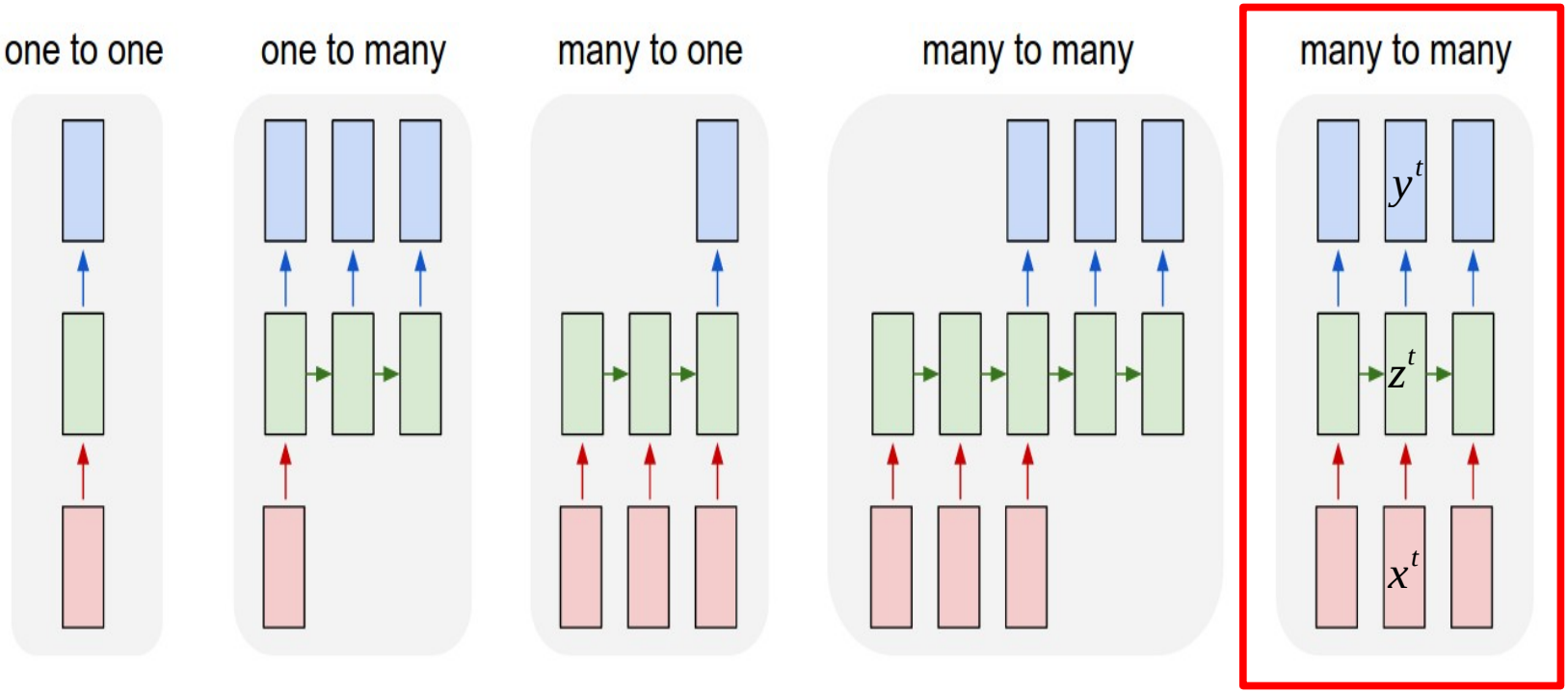


Recurrent neural networks

- Basic example for many-to-many prediction
  - Hidden state computed from current input and previous hidden state
  - Output is computed from current hidden state

$$z^t = f_z\left(W_{zx} x^t + W_{zz} z^{t-1}\right)$$

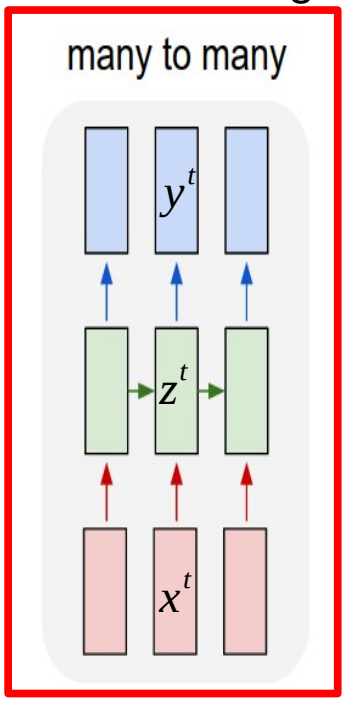
$$y^t = f_y\left(W_{yz} z^t\right)$$



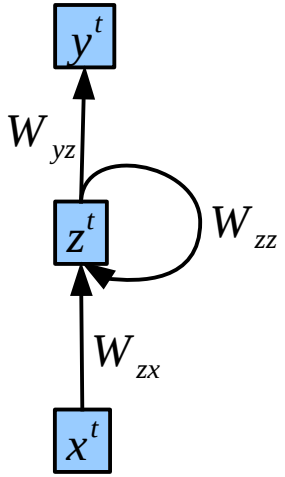
Recurrent neural network diagrams

- Two graphical representations are used

"Unfolded" flow diagram



Recurrent flow diagram



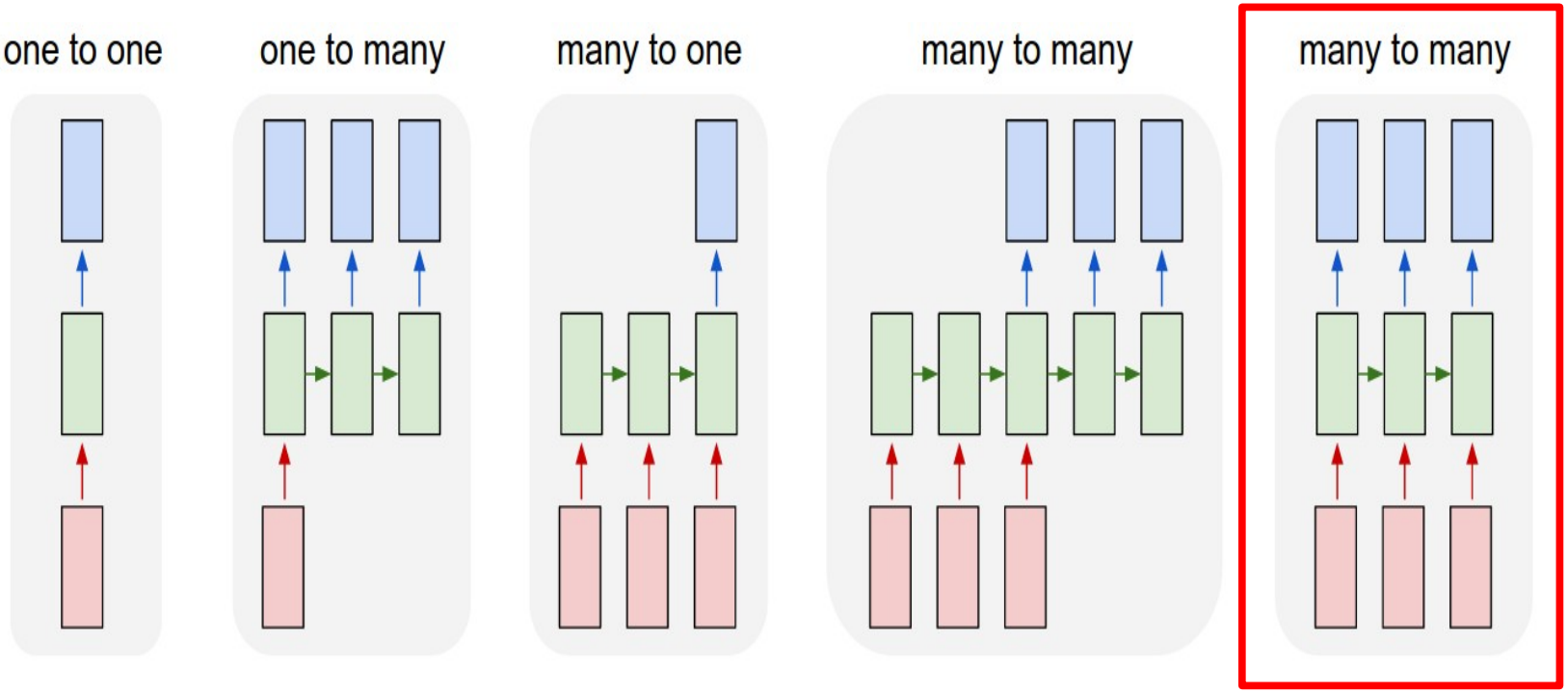
- Unfolded representation shows that we still have an acyclic directed graph
  - Size of the graph (horizontally) is variable, given by sequence length
  - Weights are shared across horizontal replications
- Gradient computation via back-propagation still works
  - Referred to as "back-propagation through 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 in translation: probability of word number t is given via softmax

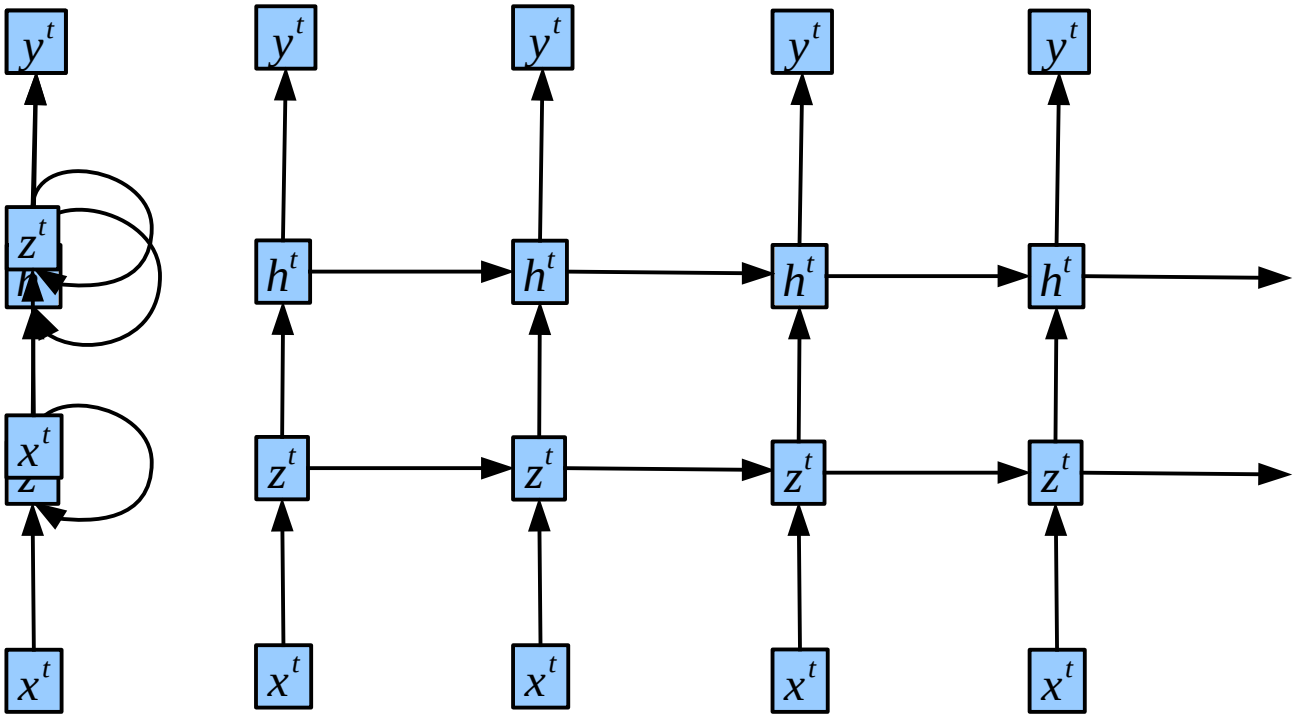
$$p(w^t = \text{dog}) = \frac{\exp y^t_{\text{dog}}}{\sum_{v=1}^{V} \exp y^t_v}$$

$$z^t = f_z(W_{zx} x^t + W_{zz} z^{t-1})$$
$$y^t = f_y(W_{yz} z^t)$$



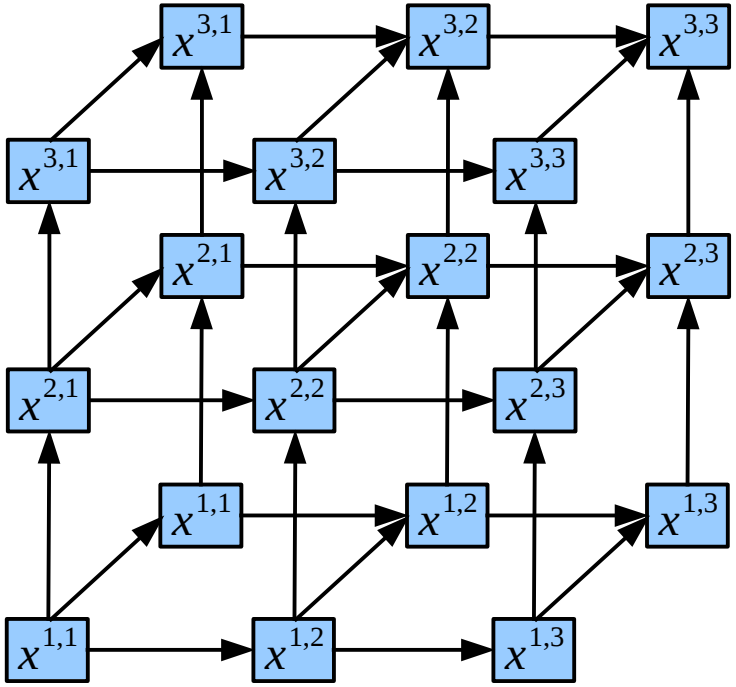
More topologies: “deep” recurrent networks

- Instead of a recurrence across a single hidden layer, consider a recurrence across a multi-layer architecture
- It's our friend: the feed-forward network



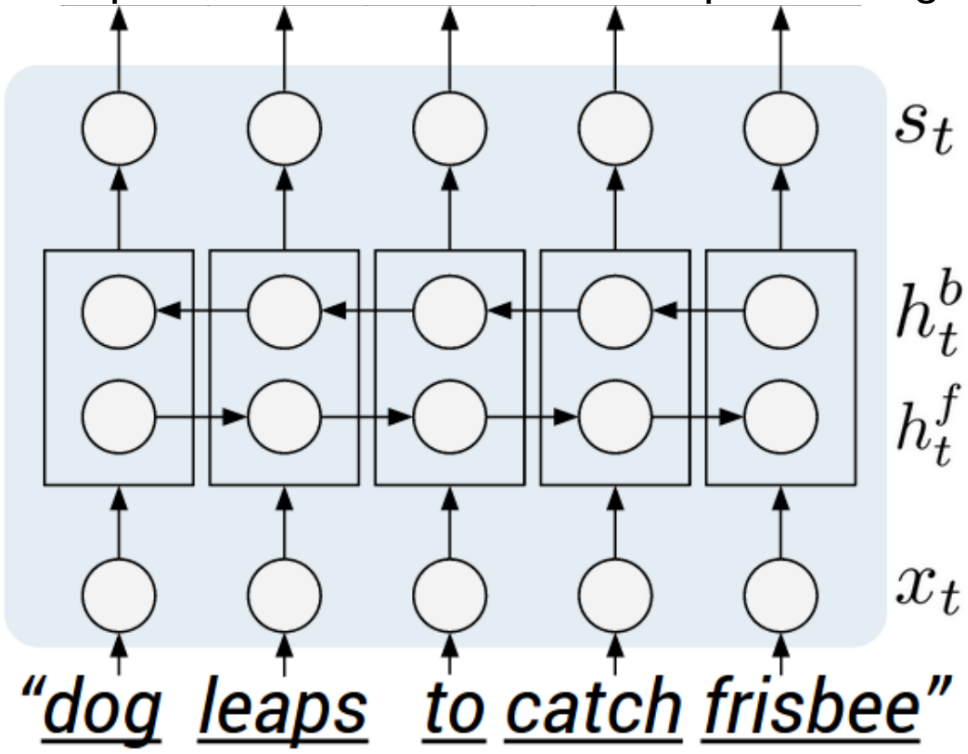
More topologies: multi-dimensional recurrent networks

- 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



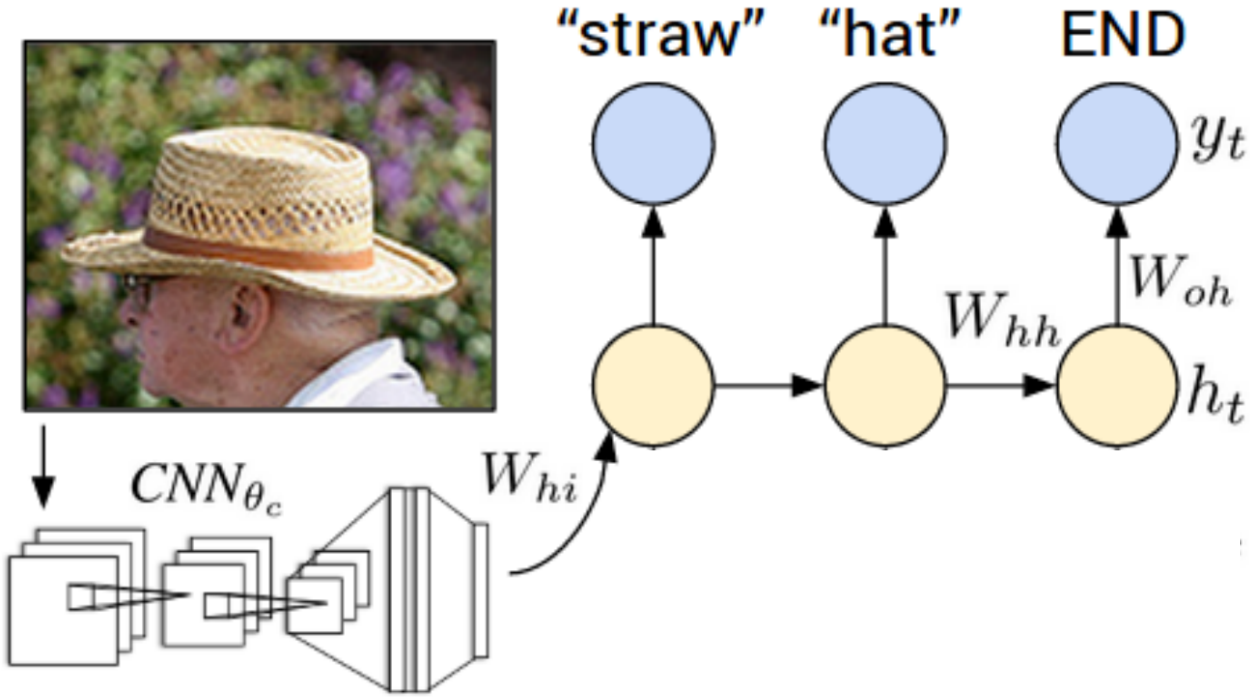
More topologies: bidirectional recurrent neural networks

- Two recurrences, one in each direction
  - Aggregate output from both nets at each time step
- Contextual representation at each time step
  - Both left and right context are take into account
- Possible on given input sequences of arbitrary length
  - Not on output sequence, since it needs to be predicted/generated



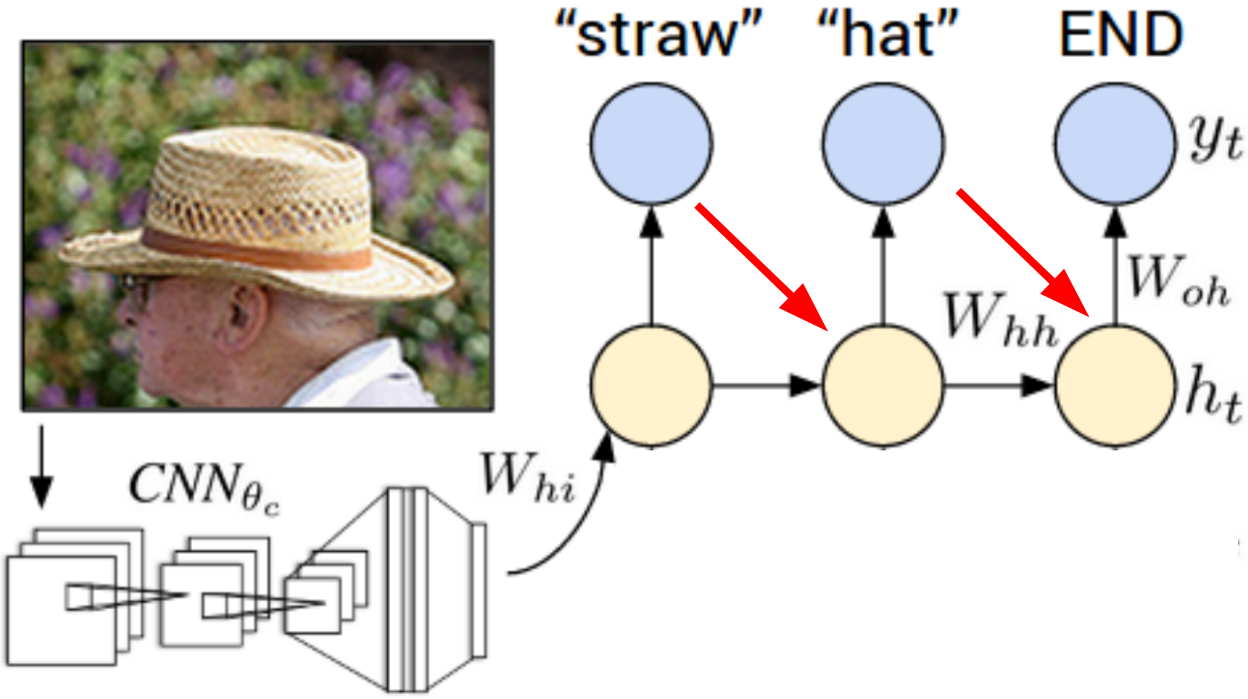
More topologies: output feedback loops

- Example of an image captioning system
- Hidden state evolution is **deterministic non-linear dynamical system**
  - Output at time t independently drawn given state at time t
  - Cannot ensure sequential coherency on output: will generate gibberish

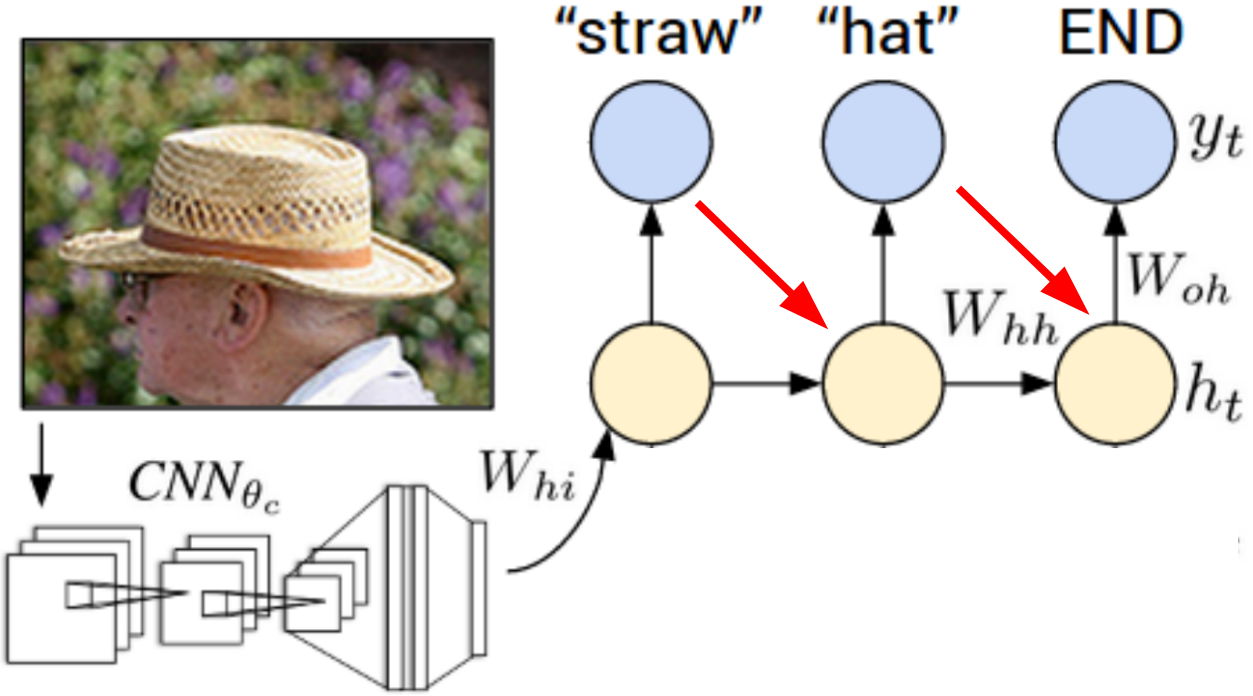


More topologies: output feedback loops

- Example of an image captioning system
- **Stochastic non-linear dynamical system over joint (state, output) pair**
  - State at time t dependent on all previous outputs
  - Output at time t therefore also, no limit on order of the Markov chain
  - Enforces sequential coherency, generates proper sentences



More topologies: output feedback loops



man in black shirt is playing guitar.

construction worker in orange safety vest is working on road.

two young girls are playing with lego toy.

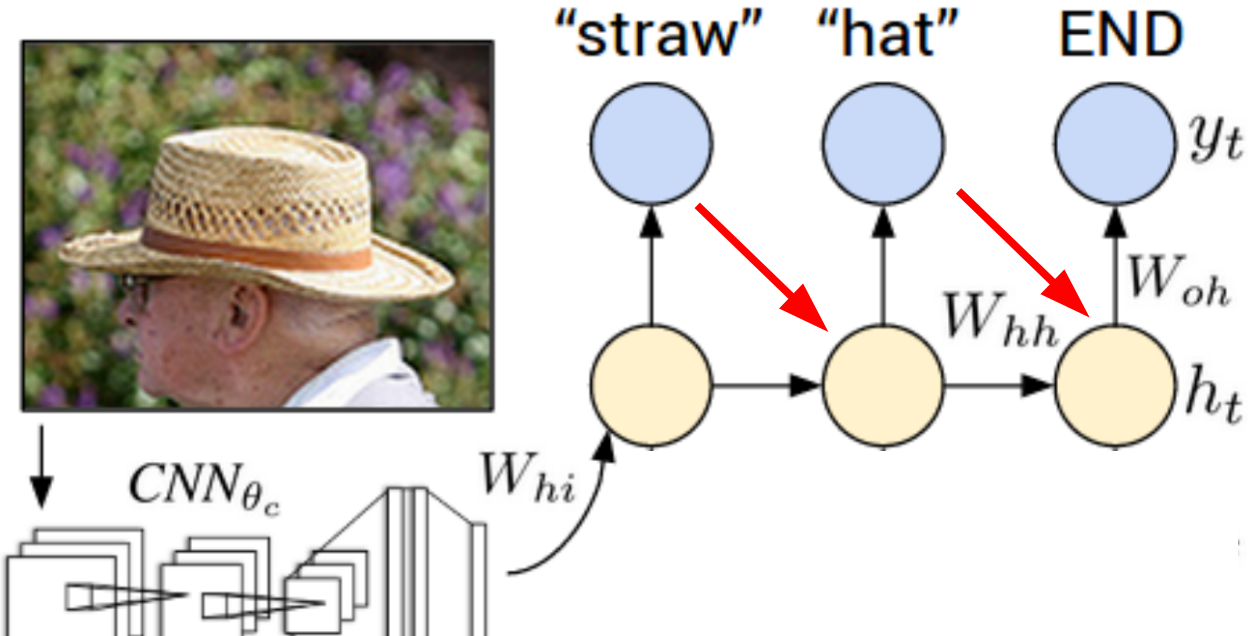
boy is doing backflip on wakeboard.

Output generation from RNN with output feedback

- Sample from distribution on next output conditioned on input and prev. input
  - Deterministic state evolution given previous outputs, and input

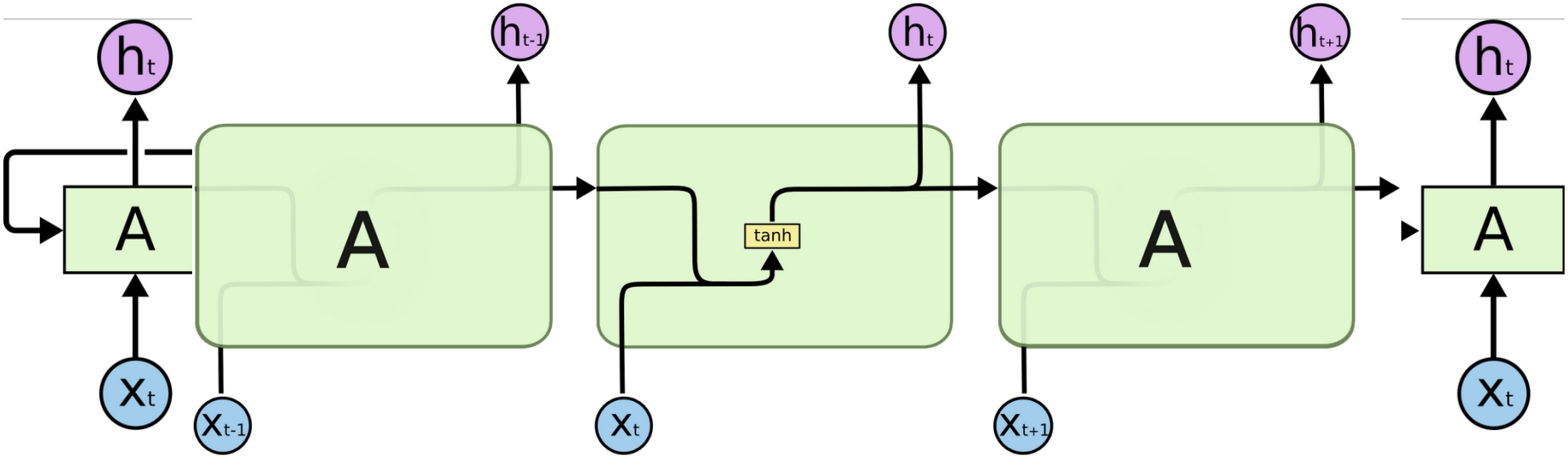
$$p(y^t | y^{1,\ldots}, y^{t-1}, x)$$

- Beam search for approximate maximum likelihood output sequence
  - Expand best k partial output sequence at current time step to k |V|
  - Keep best k for next iteration
  - Terminate paths upon “STOP” symbol, or after T steps
  - Report max likelihood sequence among k



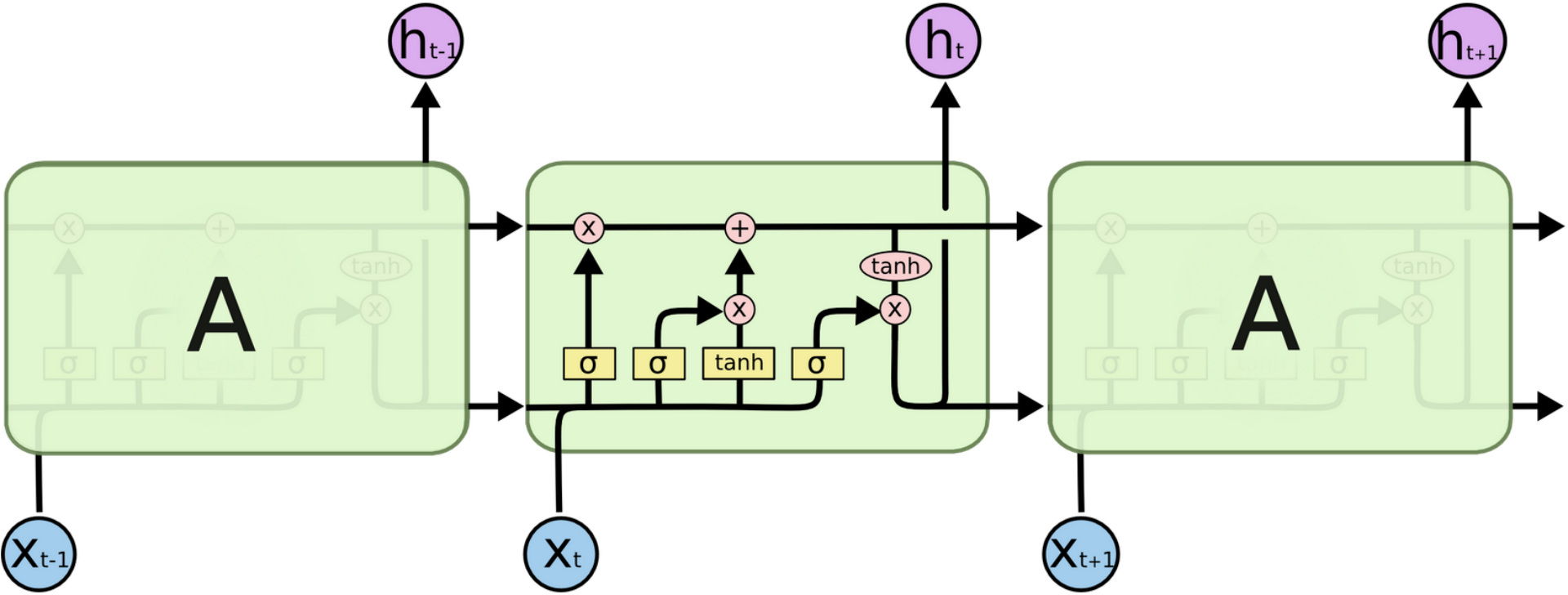
Gated recurrent units

- General feed-forward networks (incl. recurrent and conv. nets) based on
  - linear function of previous nodes
  - Point-wise non-linearity
- Input and previous state always “write” on the state in the same manner
  - Limits capacity to capture long-range dependencies (Hochreiter 91, Bengio '94)



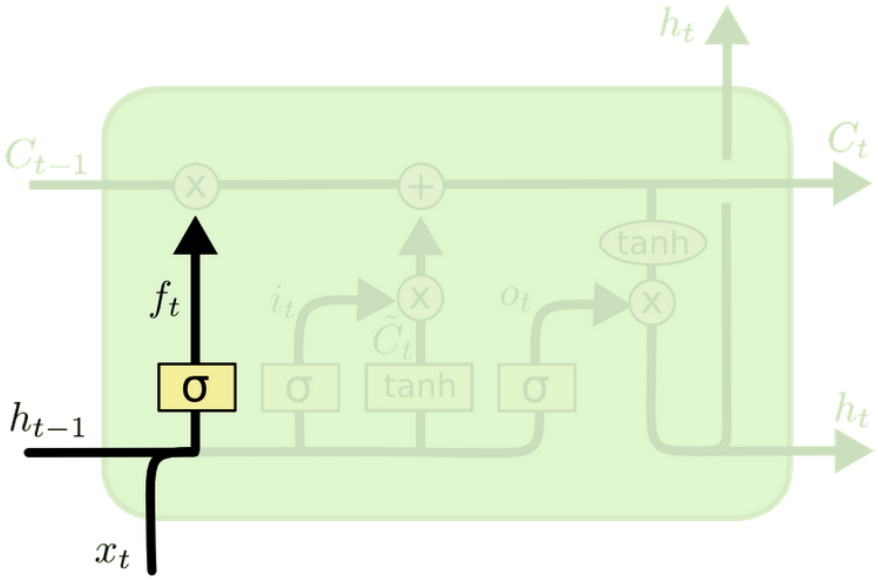
Long short-term memory (LSTM) cells

- 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



Long short-term memory (LSTM) cells

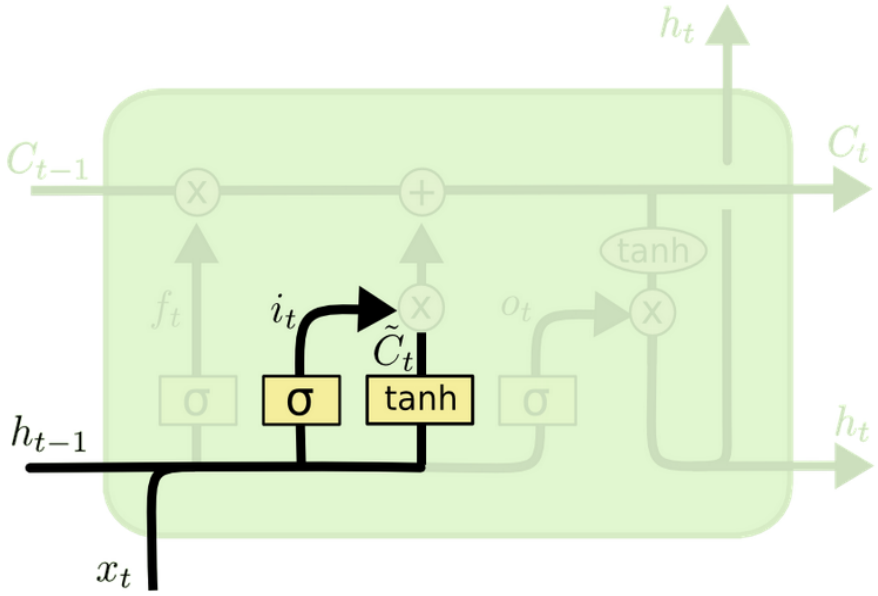
- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state h and a “memory cell” **c**
- Involves a number of additional processing elements
  - Forget gate: **f**



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

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**
  - Input modulator: $\widetilde{C}$

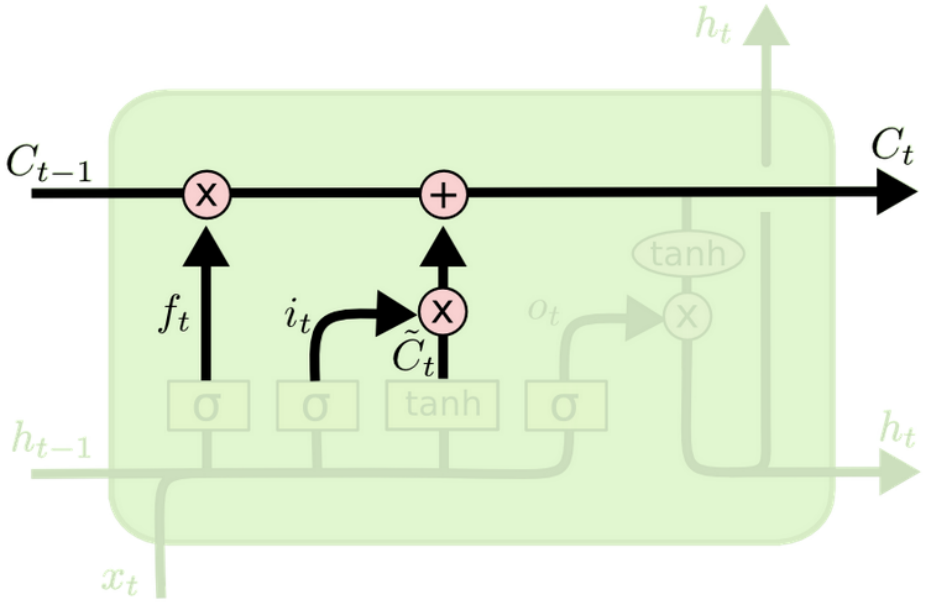


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Long short-term memory (LSTM) cells

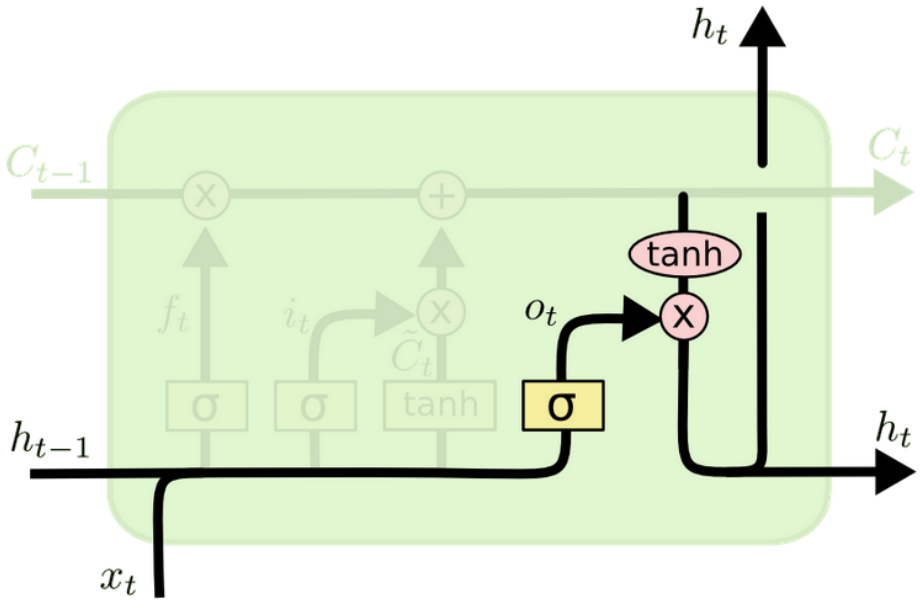
- Introduced by Hochreiter & Schmidhuber (Neural Computation, 1997)
- LSTM defines a dynamical system on hidden state h and a “memory cell” **c**
- Involves a number of additional processing elements
  - Memory cell update



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

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**
  - Hidden state update

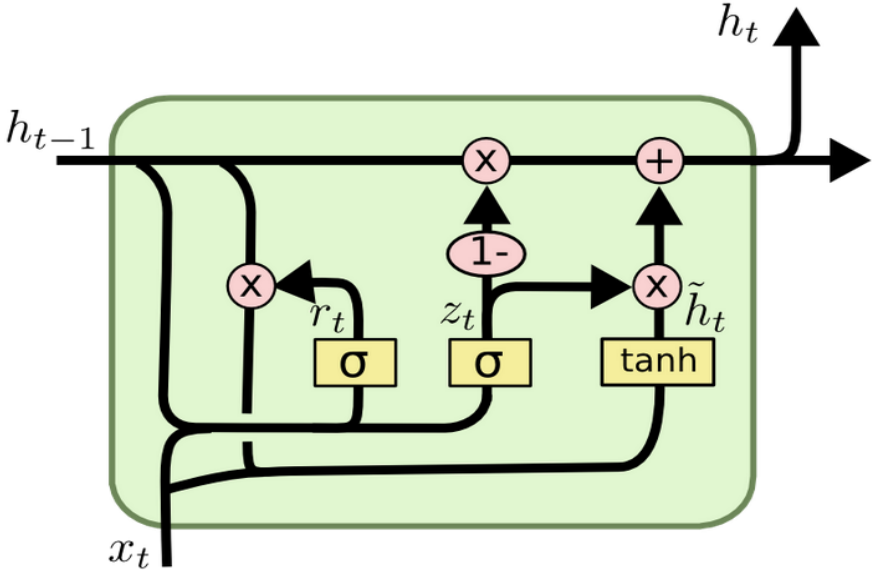


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Gated Recurrent Unit cells

- GRU is simpler model than LSTM (Cho et al., Empirical Methods in Natural Language Processing, 2014)
- Involves a number of additional processing elements
  - Forget gate: **z**
  - Read gate: **r**



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

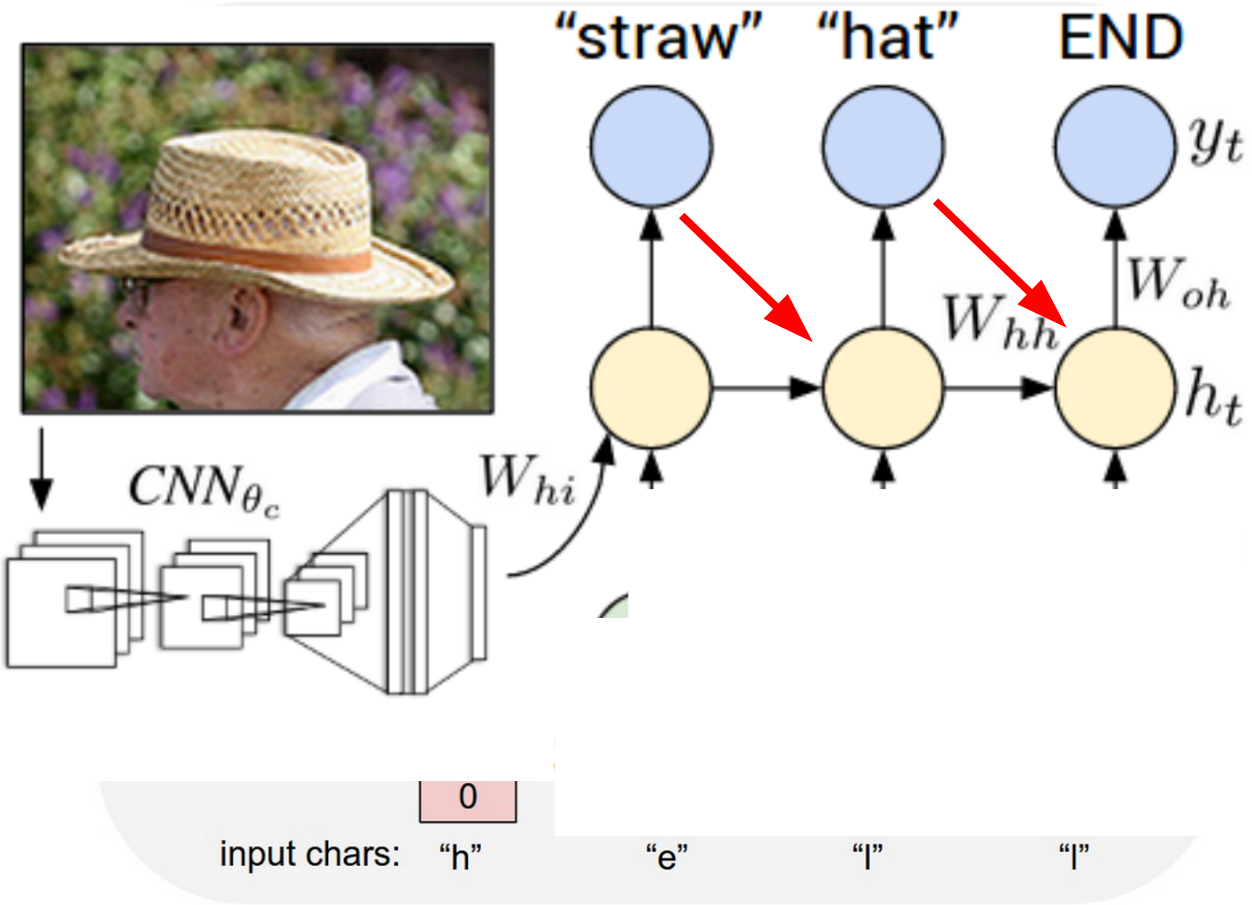
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Examples of character-level LSTM language model

- Hidden layer consists of LSTM cells
- Input: 1-of-k coding of previous character in sequence
- Training objective: log-probability of next character given past sequence



Examples taken from Andrej Karpathy

Examples of character-level LSTM language model

- 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, as well as long range grammatical dependecies

Examples of character-level LSTM language model

- Training data: all of Shakespeak (4.4 MB)
- Random sample from the trained model:

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain 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 nues 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

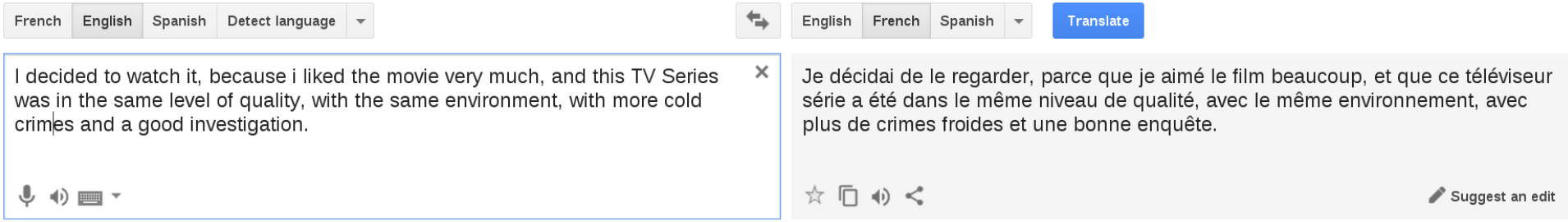
Examples of character-level LSTM language model

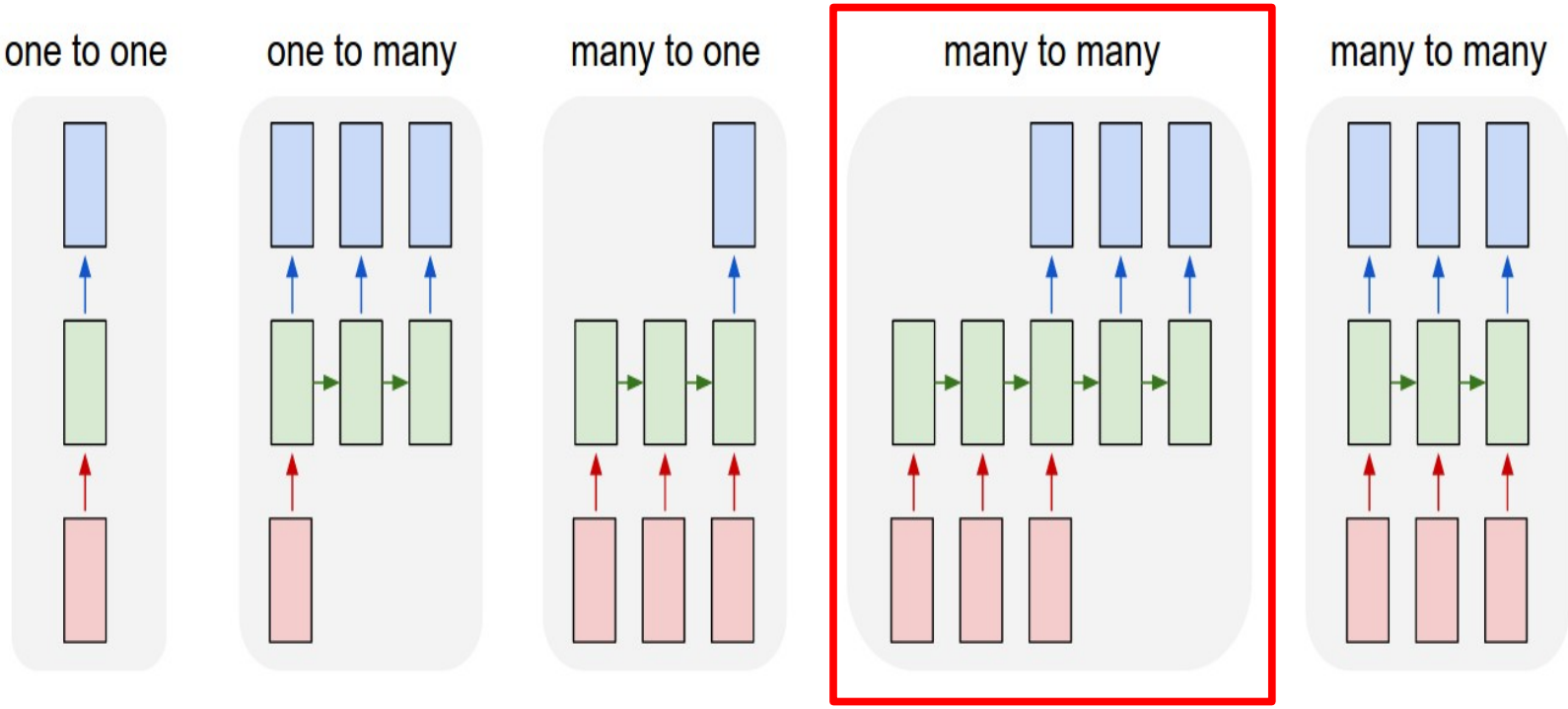
- Training data: linux source code (474 MB)
- Very long range dependencies on bracket structure

```
/*
 * 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;
}
```

Modeling sequential data

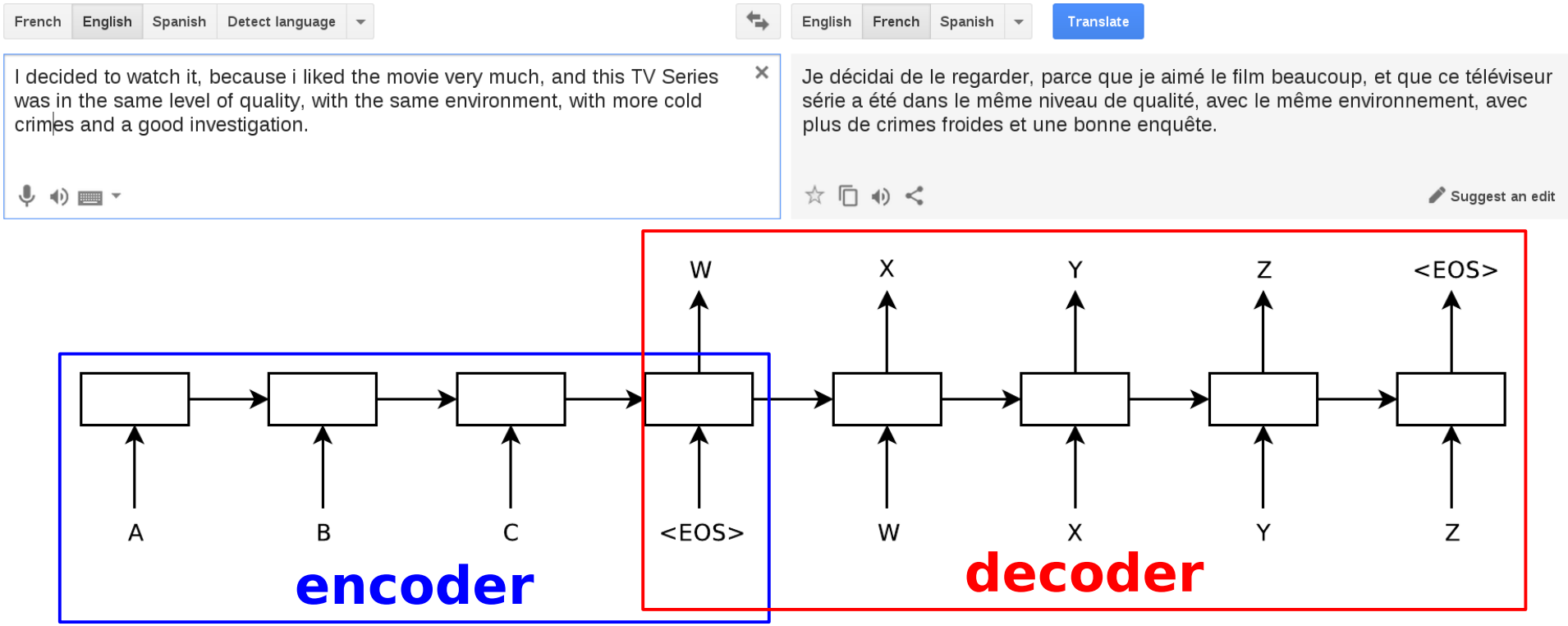
- Translation of a sentence into another language





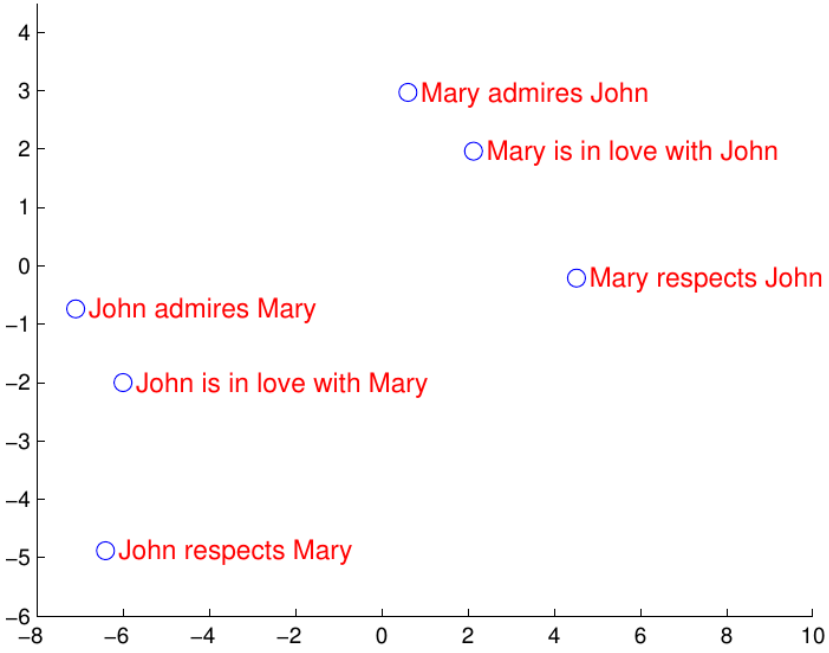
Encoder-decoder machine translation

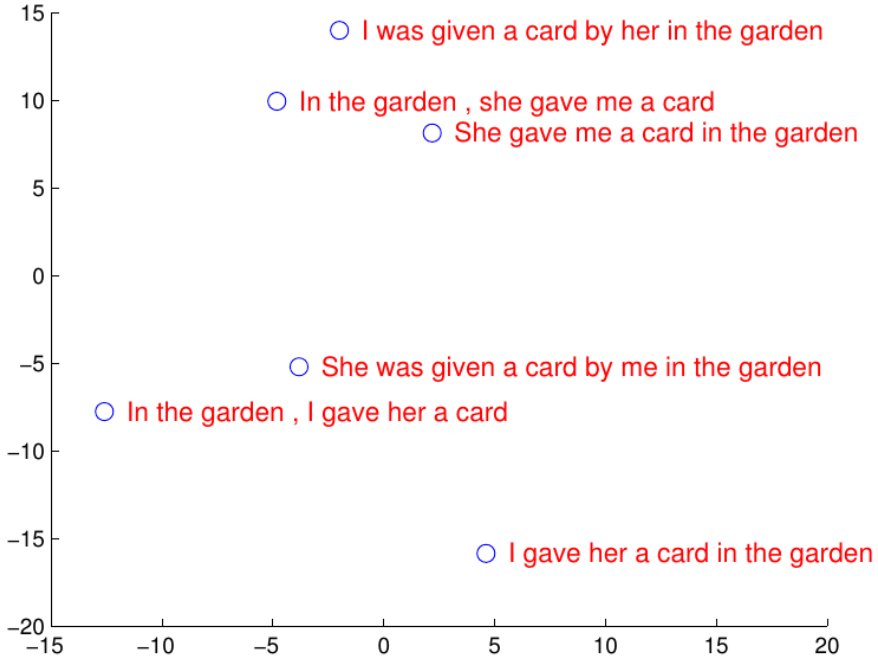
- Read source sentence with **encoder** (Sutskever et al., NIPS 2014)
  - Reading input in reverse yields better result: more short dependencies
- Generate target sentence with **decoder** RNN (LSTM, GRU, …)
  - Uses a different set of parameters
  - Uses output feedback to ensure output coherency



Encoder-decoder machine translation

- **Encoder** and **decoder** can be learned on multiple language pairs in parallel
  - (English to French) and (German to French) use same decoder
  - (English to French) and (English to German) use same encoder
- PCA projection of LSTM state after reading several phrases
  - Word order very important for meaning, captured in state clusters





Summary of recurrent networks

- “Unfolding” recurrent networks allows to recognize them as feedforward networks with weight-sharing across recurrent connections
- Recurrent networks are powerful tools to model sequential data
  - Sequential input and/or sequential output
  - Input and output sequence may be aligned or not
- Recurrent networks over a 1-dimensional time axes may be generalized
  - To multidimensional structures data on input or output
  - To encoder-decoder networks
  - Deep recurrent networks
- Recurrent networks with a gating mechanism are found to be much stronger in practice 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
- Impressive results on natural language processing tasks, including
  - Image captioning
  - Machine translation

Further reading

- “Pattern Recognition and Machine Learning”

  Chris Bishop.

  Springer, 2006.

- “Deep Learning”

  Ian Goodfellow, Yoshua Bengio, Aaron Courville.

  MIT Press, in preparation.

  http://www.deeplearningbook.org/