Machine Intelligence 1

1.5 Deep Learning

Prof. Dr. Klaus Obermayer

Fachgebiet Neuronale Informationsverarbeitung (NI)

WS 2017/2018

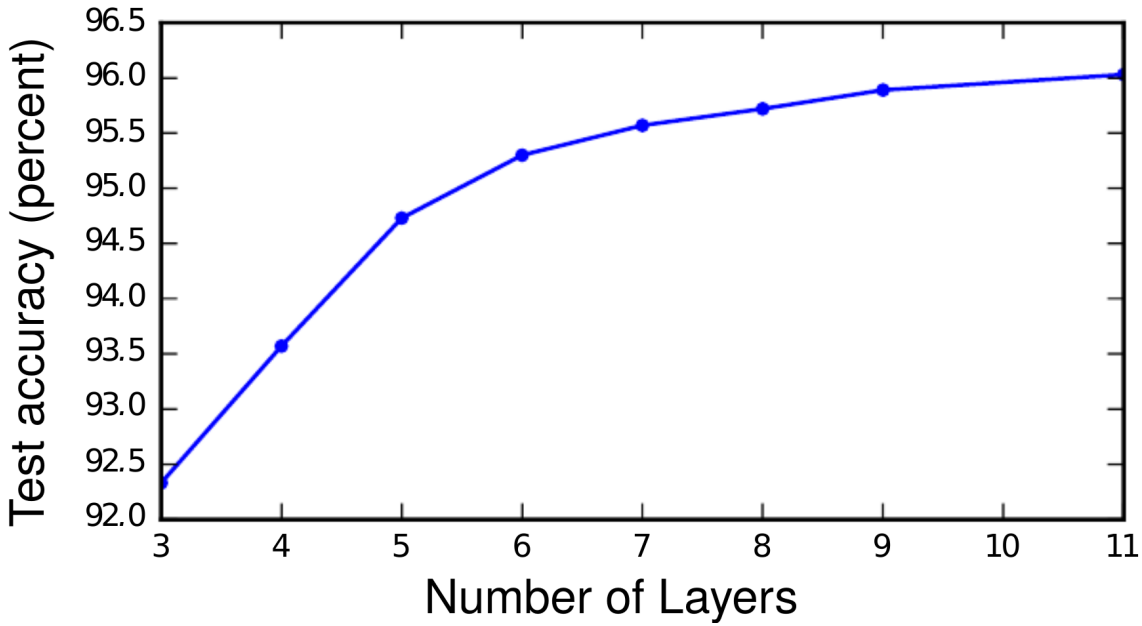
1.5.1 Deep Neural Networks

Shallow and deep networks

- deep MLP were long thought to be **un-trainable**
  - lower layers need to converge first $\rightsquigarrow$ long training times
  - sigmoid transfer functions f have *vanishing gradients*, i.e. if $|h_i^v|$ is large, $f'(h_i^v)$ is very small $\rightsquigarrow$ slow convergence
  - large number of parameters $\rightsquigarrow$ available datasets too small
- shallow architectures appeared to be the a reasonable choice
  - MLP with 1 hidden layer are *universal function approximators*
- *support vector machines* dominated ML by the end of the 1990's
- research was practically abandoned in the early 2000's

Deep neural networks

- number of layers increases the performance of MLP
  - large house-number recognition data-set on street-view images
  - various architectures were trained with many tricks-of-the-trade

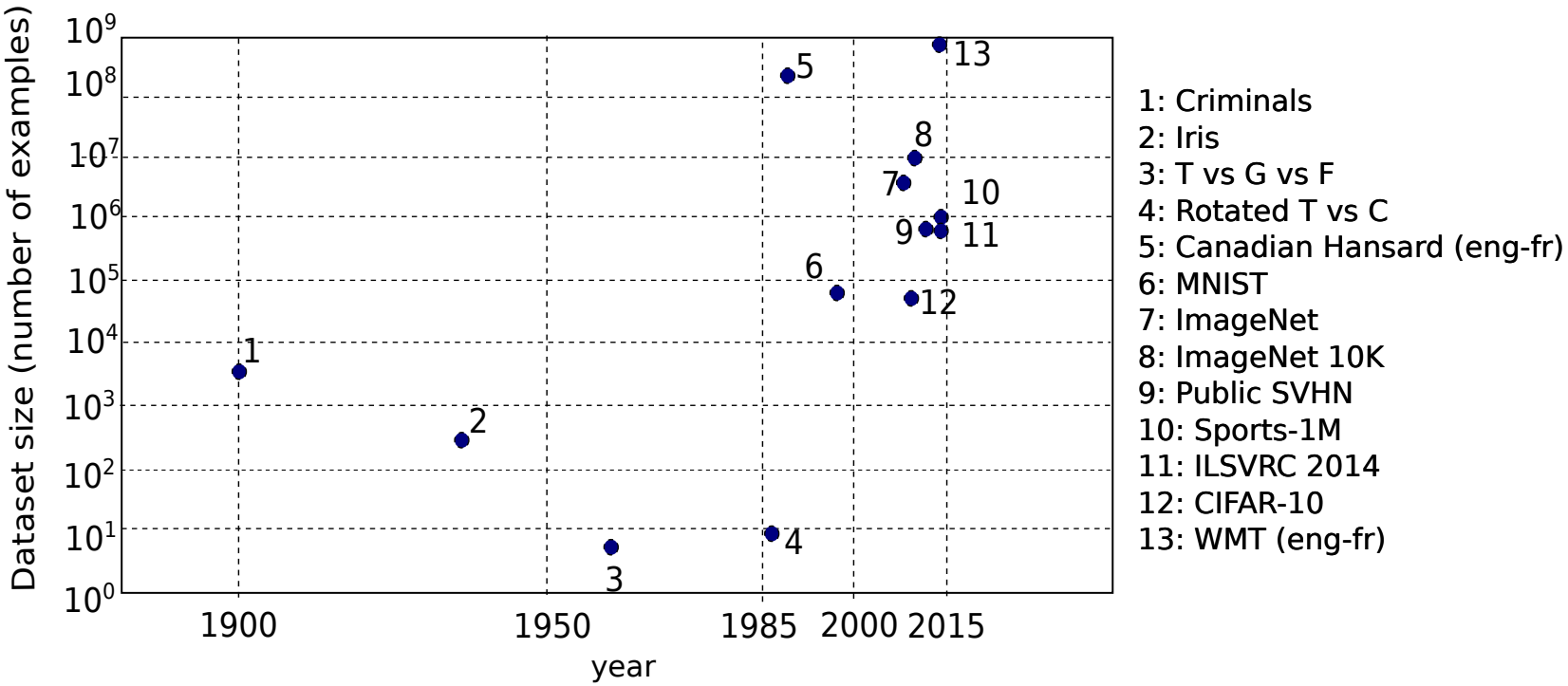


- since 2006 deep MLP won most machine learning challenges

but why?

(figure from Goodfellow et al., 2014, 2016)

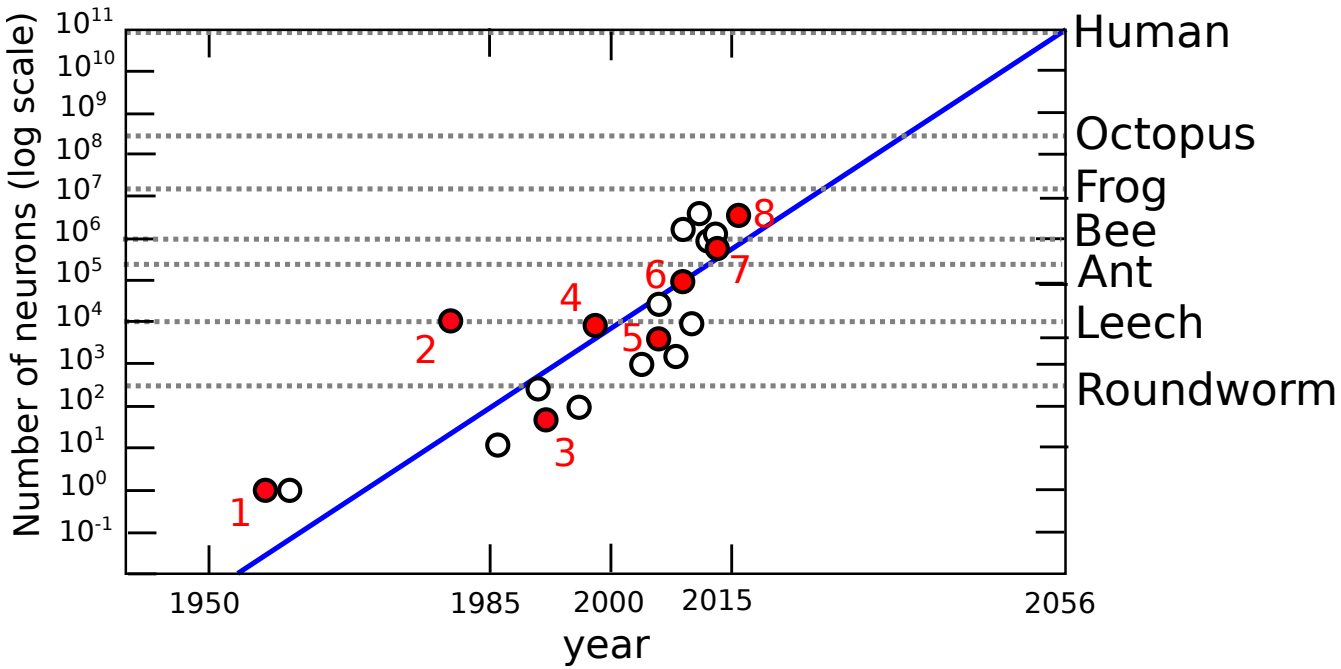
Explanation 1: training with larger data sets



(graph adapted from Goodfellow et al., 2016)

Explanation 2: increasing computation power

- GPU-clusters update up to a billion parameters now

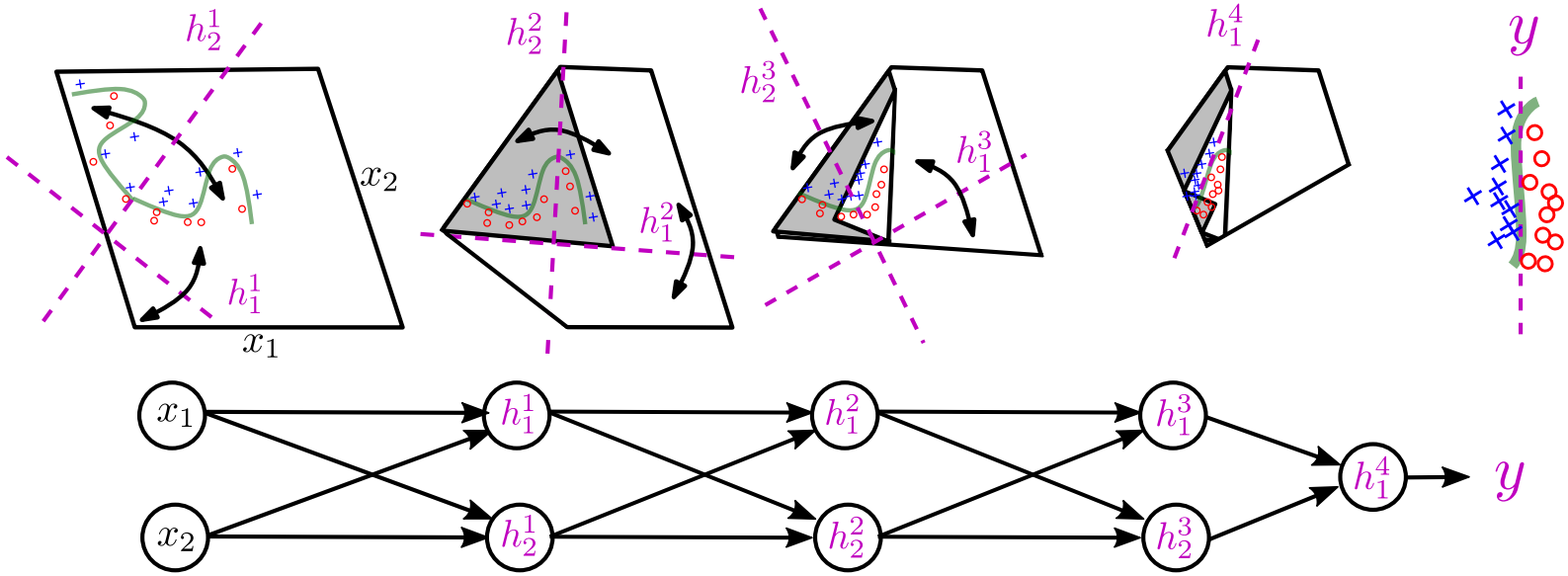


1.	Perceptron (Rosenblatt, 1962)	5.	Deep belief network (Hinton and Salakhutdinov, 2006)
2.	Neocognition (Fukushima, 1980)	6.	GPU accelerated DBN (Raina et al., 2009)
3.	MLP for speech recognition (Bengio et al., 1992)	7.	Multi-GPU conv. net (Krizhevsky et al., 2012)
4.	LeNet-5 (LeCun et al., 1998)	8.	GoogLeNet (Szegedy et al., 2015)

(graph adapted from Goodfellow et al., 2016, data fom Wikipedia)

Explanation 3a: deep networks can exploit invariances

- example: deep network for classification
  - classification boundary exhibits symmetries/invariances



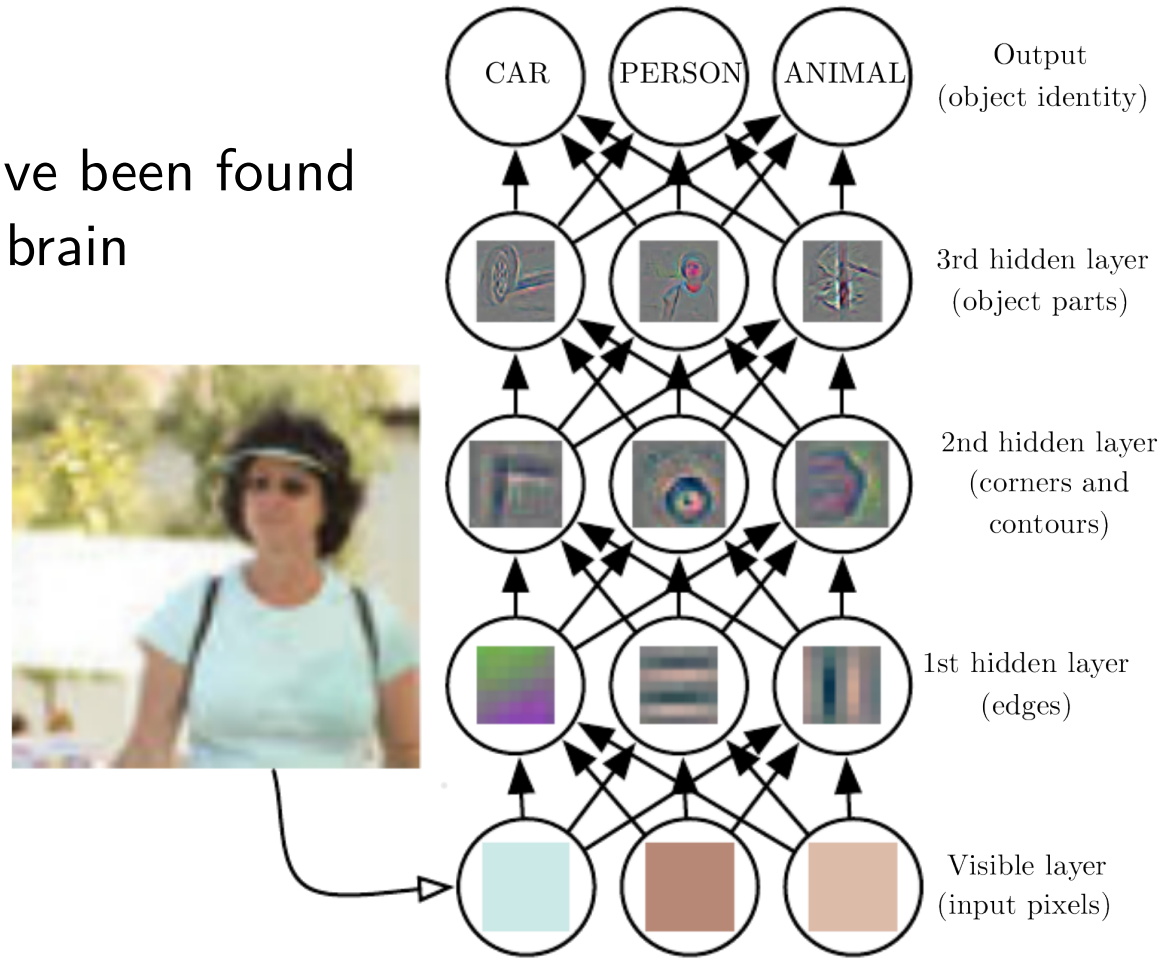
- non-linear transfer functions induce invariances in the output
  - $f(h_i^v) = |h_i^v|$ "folds" input space along the learned hyperplane h_i^v

$$h_i^1 = \sum_{j=1}^{2} w_{ij}^1 x_j + b_i^1 , \qquad h_i^v = \sum_{j=1}^{2} w_{ij}^v f(h_j^{v-1}) + b_i^v , \qquad y(\underline{\mathbf{x}}) = \text{sign}(h_1^4)$$

(figure adapted from Montúfar et al., 2014)

Explanation 3b: deep representations

- deep MLP can encode *representations* of increasing complexity
- similar representations have been found in the visual areas of the brain
- potentially better *generalization* than shallow representations



(figure from Goodfellow et al. (2016), based on data from Zeiler and Fergus (2014))

Explanation 4: improved tricks-of-the-trade

- *rectified-linear* transfer functions

$$f(h_i^v) = \max(0, h_i^v) \Rightarrow f'(h_i^v) = \begin{cases} 0 & , h_i^v \leq 0 \\ 1 & , h_i^v > 0 \end{cases}$$

- efficient *GPU computation* and *mini-batch* optimization
  $\rightsquigarrow$ Section 1.5.2

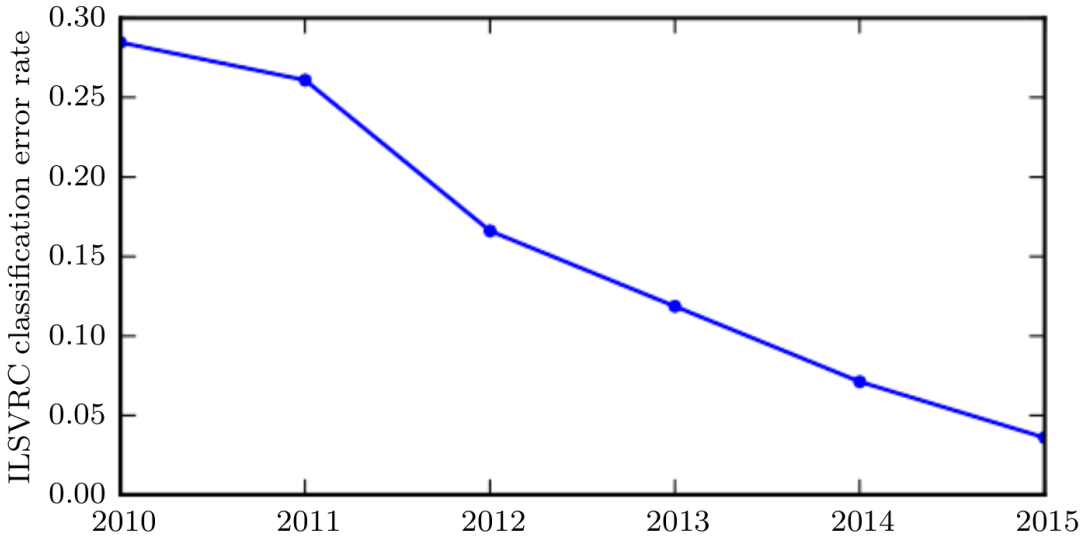
- layer-wise auto-encoder *pre-training* to initialize parameters
  $\rightsquigarrow$ Section 1.5.2

- advanced regularization methods like *dropout*
  $\rightsquigarrow$ Section 1.5.3

- deep network architectures like
  - convolutional neural networks (CNN) $\rightsquigarrow$ Sections 1.5.4 and 1.5.5
  - recurrent neural networks (RNN) $\rightsquigarrow$ Section 1.6

Result: increasing performance

- performance of deep neural networks improves continuously



(ImageNet Large Scale Visual Recognition Challenge, from Goodfellow et al., 2016)

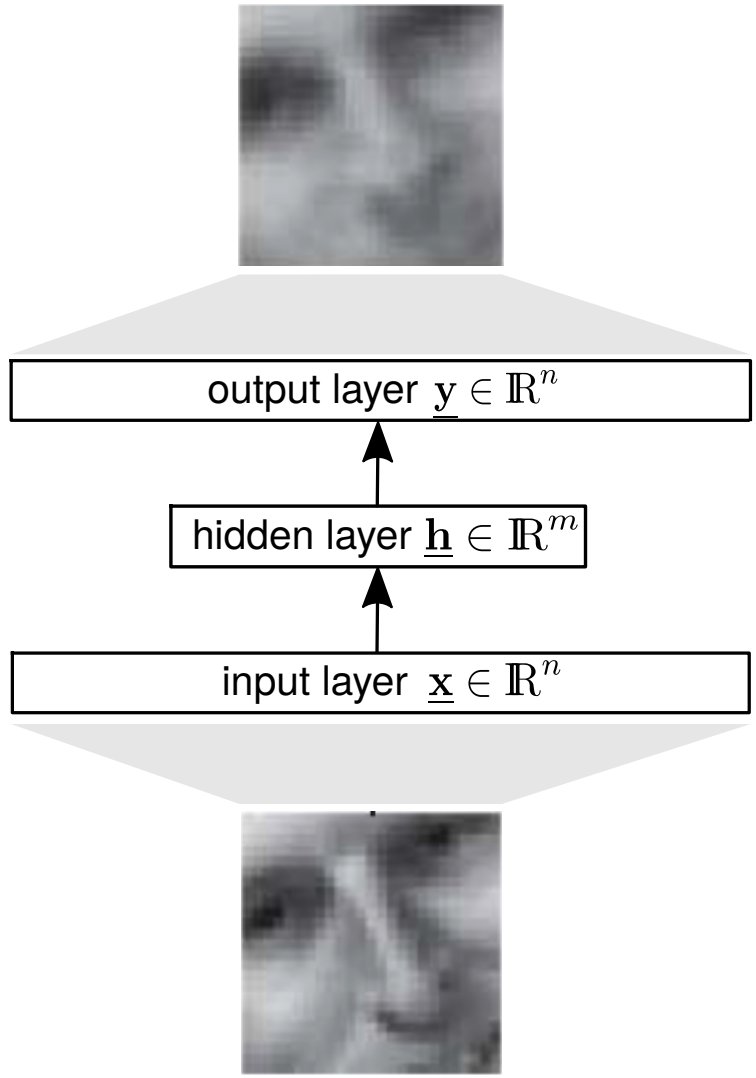
1.5.2 Auto-encoder and Pre-training

Auto-encoders

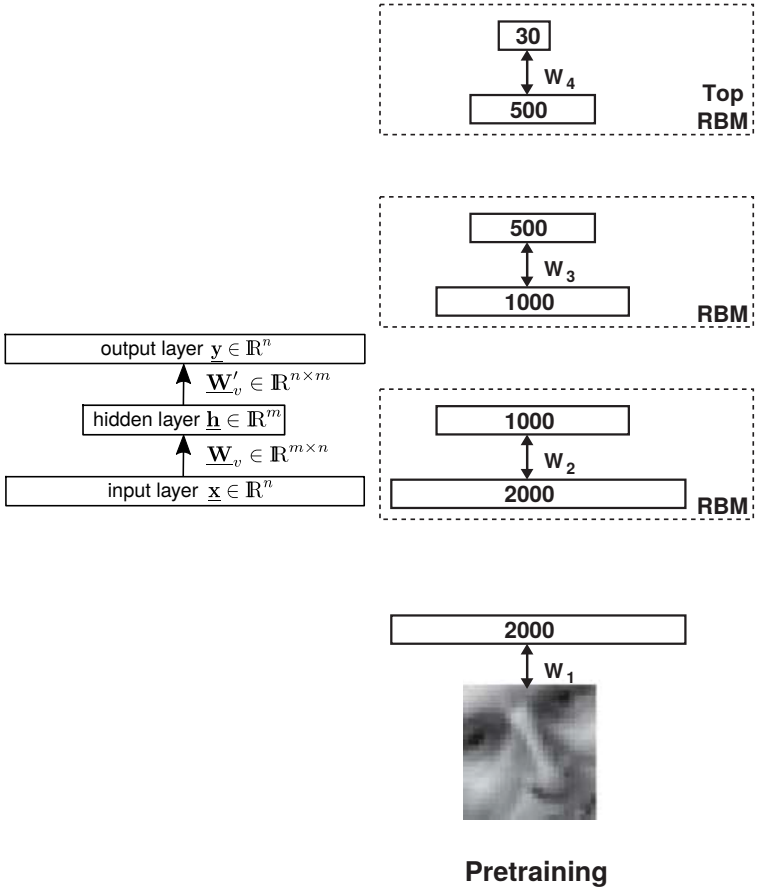
- shallow "auto-encoder" to learn invariances/representations:
  - MLP output predicts input
  - smaller hidden layer, i.e. $m \ll n$

$$E^T_{[\underline{\mathbf{w}}]} = \frac{1}{p}\sum_{\alpha=1}^{p}\frac{1}{n}\sum_{i=1}^{n}\left(y_i(\underline{\mathbf{x}}^{(\alpha)}|\underline{\mathbf{w}}) - x_i^{(\alpha)}\right)^2$$

- hidden layer $\underline{\mathbf{h}}$ learns low-dimensional representation/encoding of input
- compact representations often *generalize* better to unseen inputs
- information bottleneck



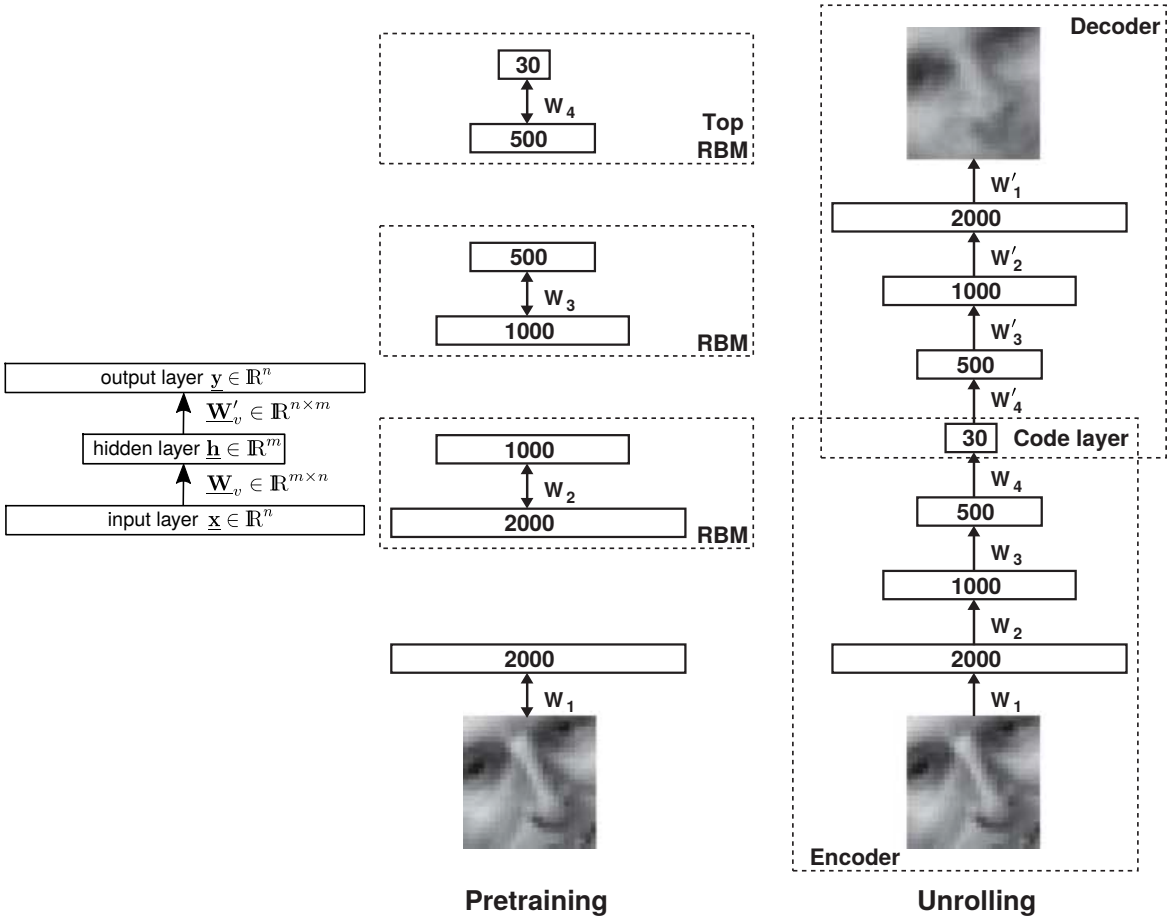
Deep belief networks (DBN)



- initialize parameters by layer-wise unsupervised pre-training
- hidden units of each layer are the input to the next

(Hinton and Salakhutdinov, 2006, used originally *Restricted Boltzmann Machines*)

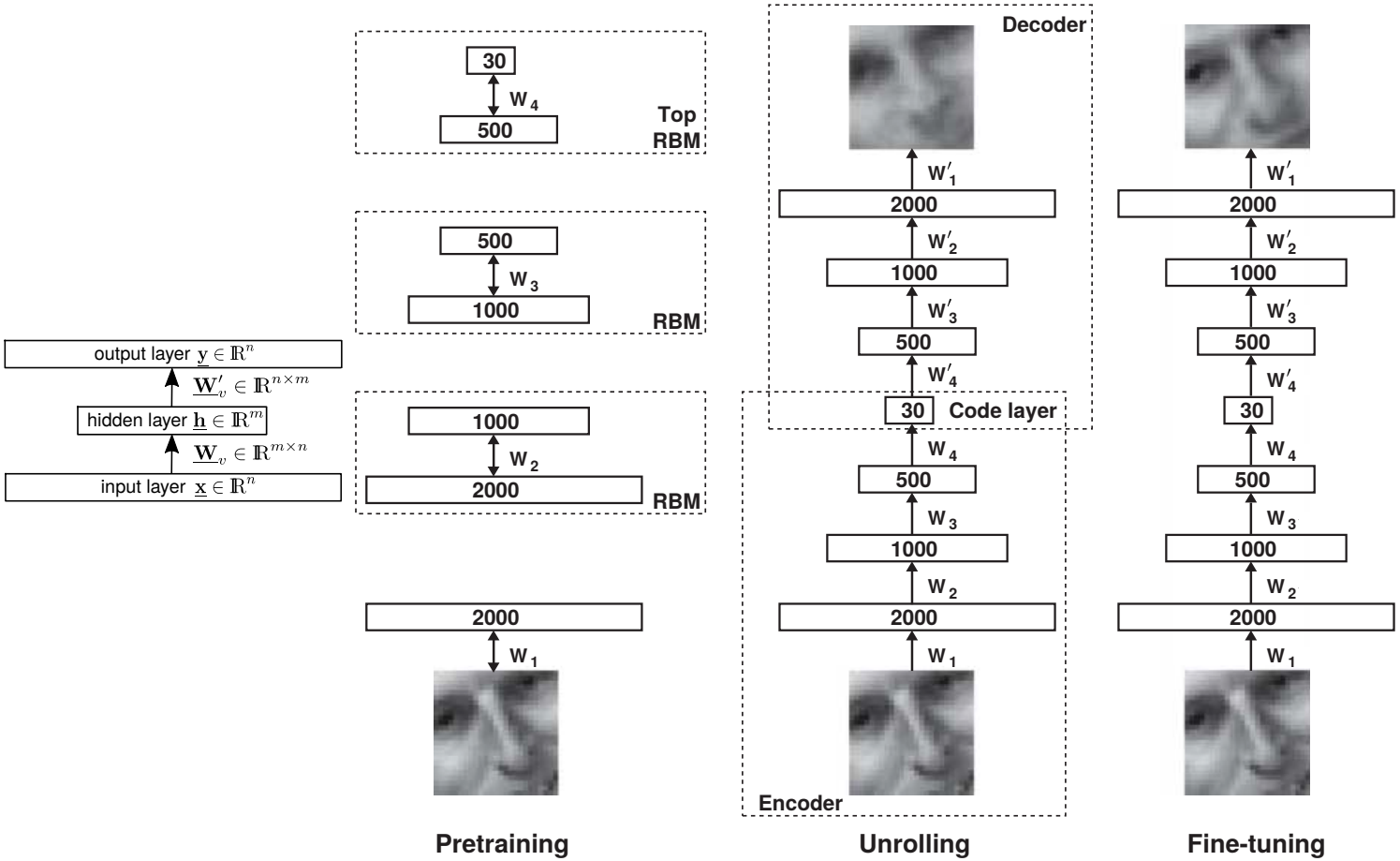
Deep belief networks (DBN)



- **encoding**: compresses input to low-dimensional representation
- **decoding**: generates an input "image" from the representation

(decoding improves with more layers, see Hinton and Osindero, 2006)

Deep belief networks (DBN)



- **generative auto-encoder** can be fine-tuned by back-propagation
- learns an efficient representation of training set plus noise ε_i

(decoding improves with more layers, see Hinton and Osindero, 2006)

Mini-batch stochastic gradient descend

- deep learning requires large training sets
  - so many samples (e.g. images) may not fit into memory
  - batch gradient descend is not feasible
- online stochastic gradient descend is very slow
  - stochasticity of update direction necessitates small learning rate η
- stochastic gradient descend with *mini-batches*
  - asynchronously request small random subsets of the data
  - update based on the average gradient of the mini-batch
  - faster convergence than online learning

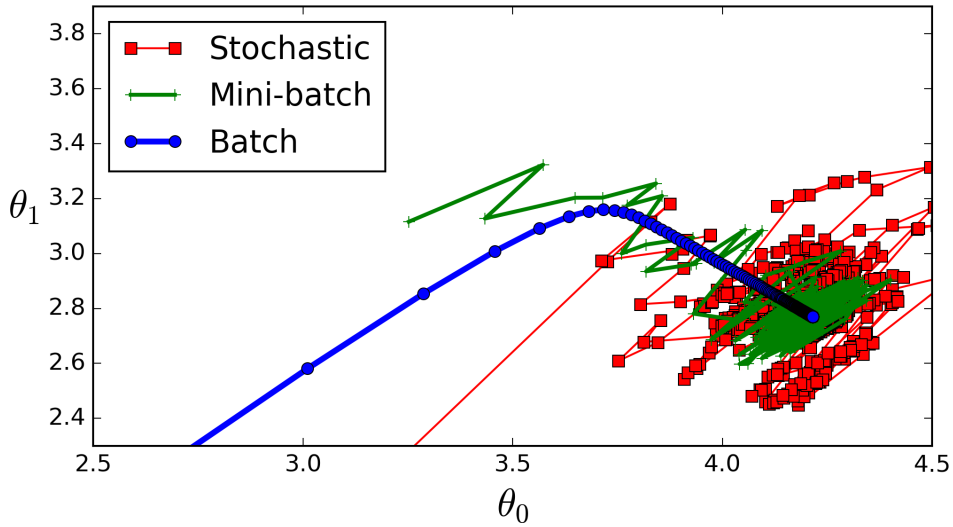
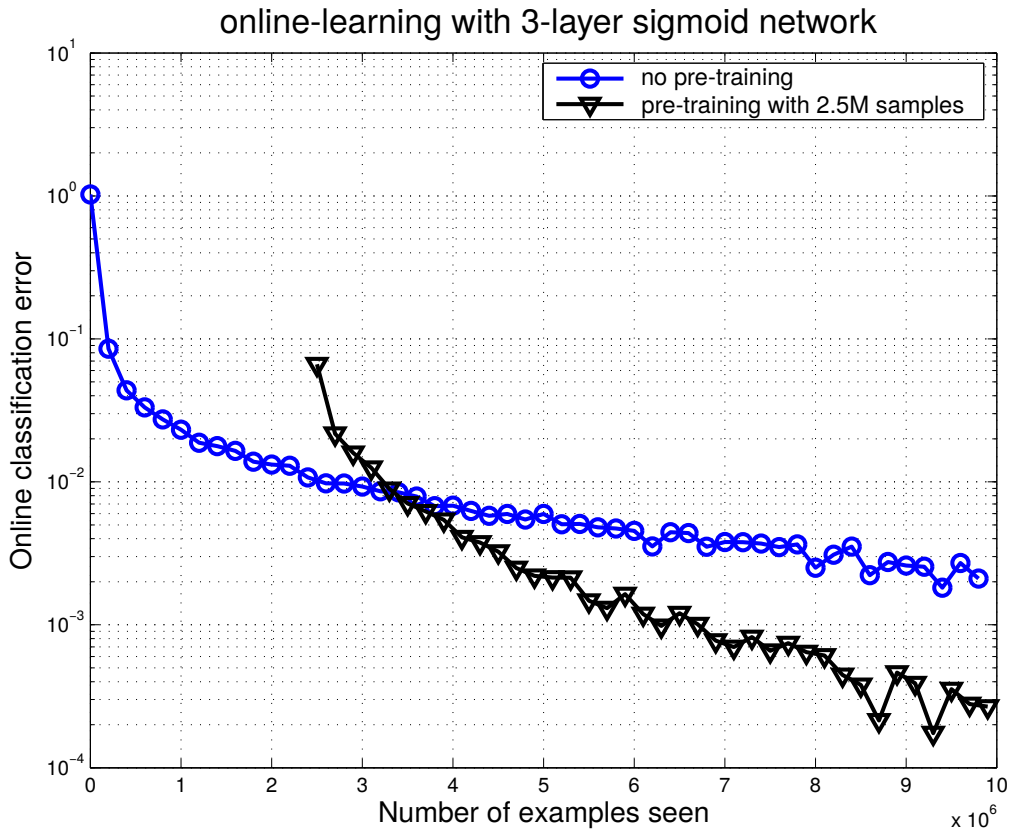


image from https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

Layer-wise pre-training

- deep MLP with random initializations require very large training sets
- layer-wise auto-encoder pre-training reduces the number of samples



(stream of randomly translated, rotated and scaled MNIST digits, Bengio, 2009)

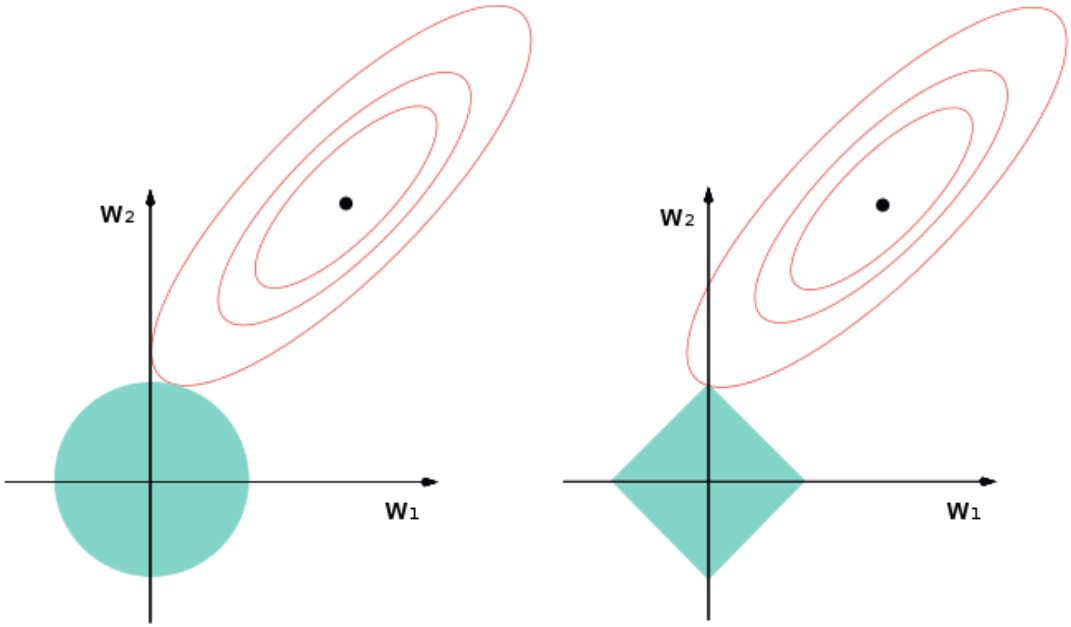
1.5.3 Regularization for Deep Learning

Norm regularization

- all previously discussed regularization methods work, e.g.

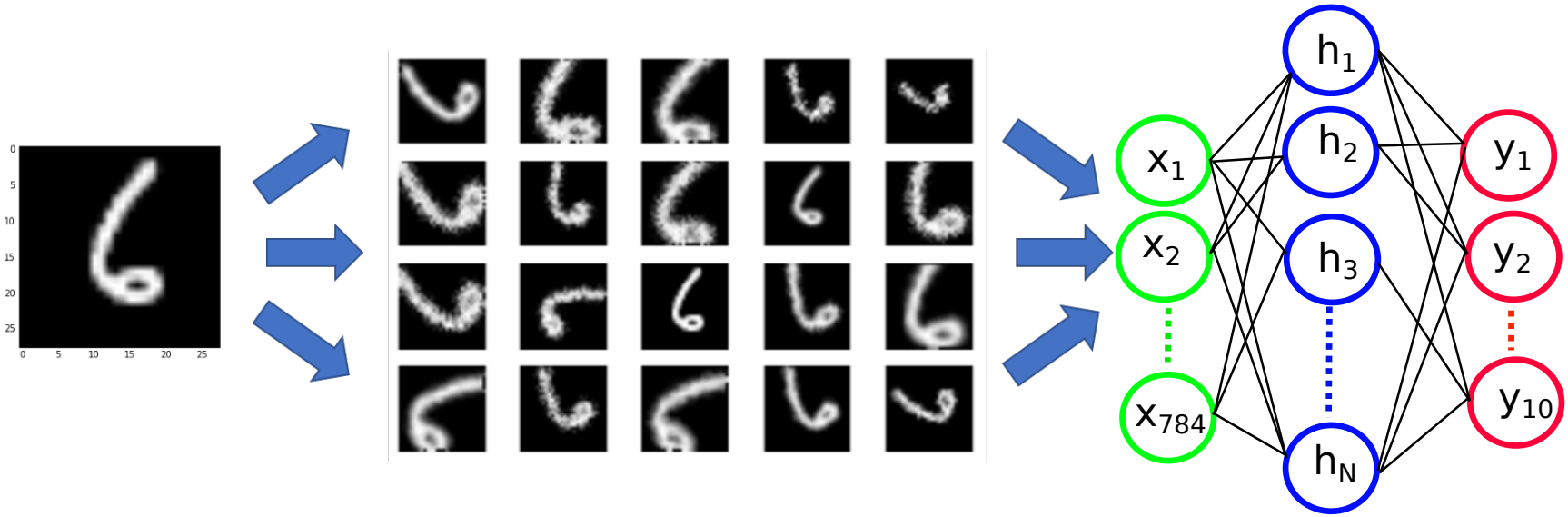
$$R_{[\underline{\mathbf{w}}]} = \sum_{i=1}^{N} w_i^2 \qquad (L_2 \text{ regularization})$$

$$R_{[\underline{\mathbf{w}}]} = \sum_{i=1}^{N} |w_i| \qquad (L_1 \text{ regularization})$$



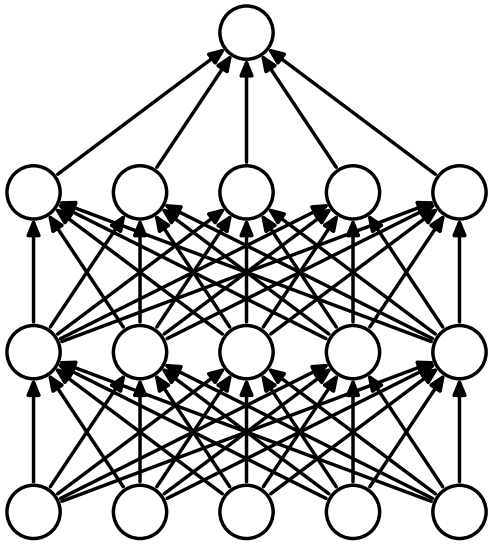
Data augmentation

- perform transformations on the training data

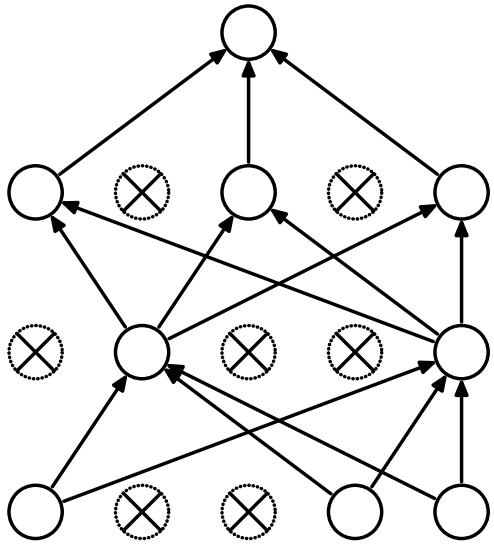


- injecting noise in training samples and labels
- output becomes *invariant* against transformations and noise

Dropout



standard neural net

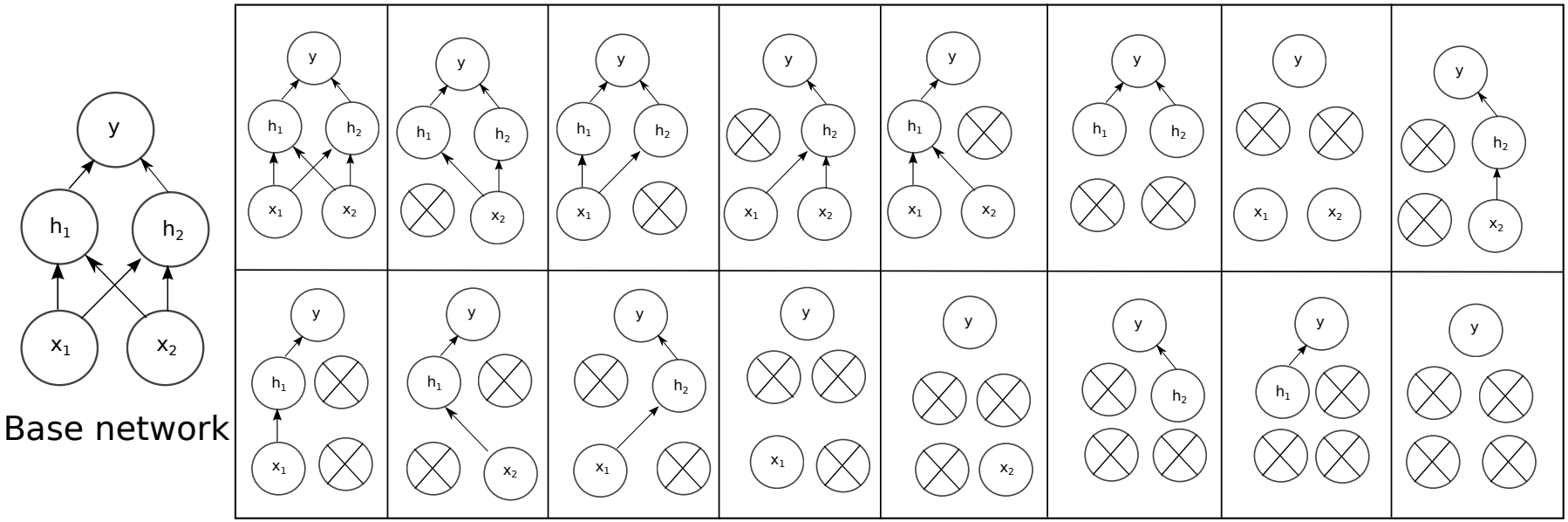


after applying dropout

- each update step deactivates a number of randomly chosen neurons
- prediction uses all neurons with proportionally scaled weights

(Srivastava et al., 2014)

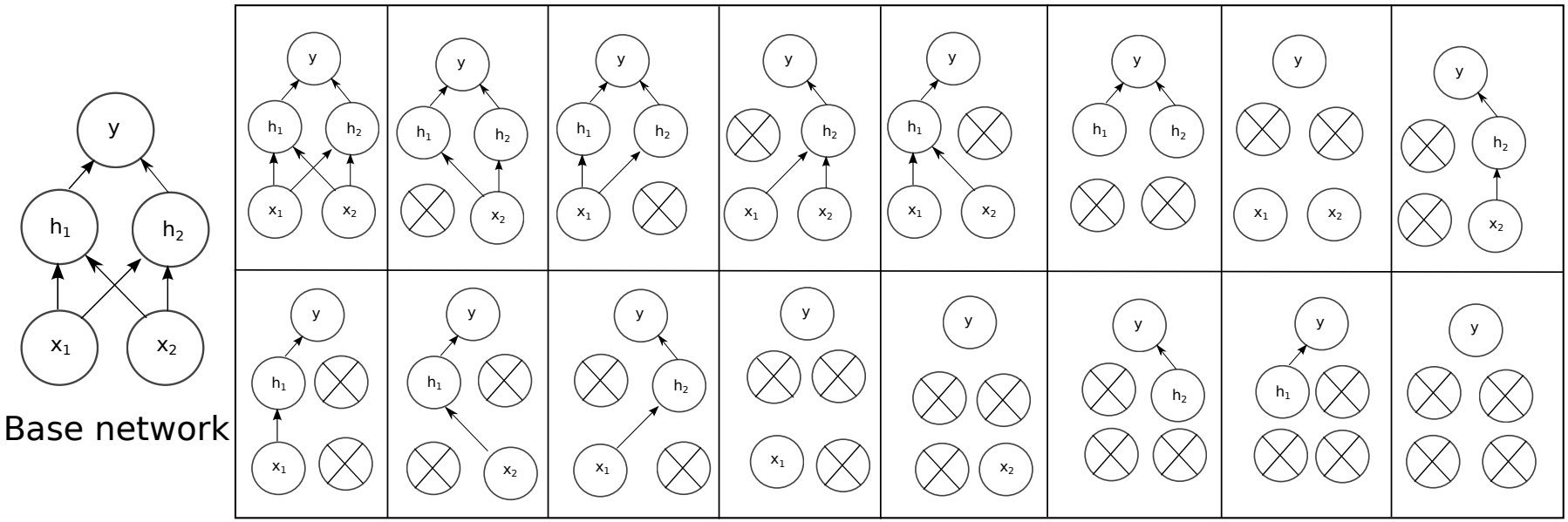
Dropout as an ensemble method



- dropout updates an ensemble of 2^n models
  - all models share parameters $\underline{\mathbf{W}}$
  - neurons i are deactivated by drawing $d_i \in \{0, 1\}$ (Bernoulli i.i.d.)
  - each neuron's output $S_i(\underline{\mathbf{x}})$ is multiplied by $d_i \in \{0, 1\}$
  - each model can be identified by a unique vector $\underline{\mathbf{d}} \in \{0, 1\}^n$

(see Goodfellow et al., 2016)

Dropout as an ensemble method



Base network

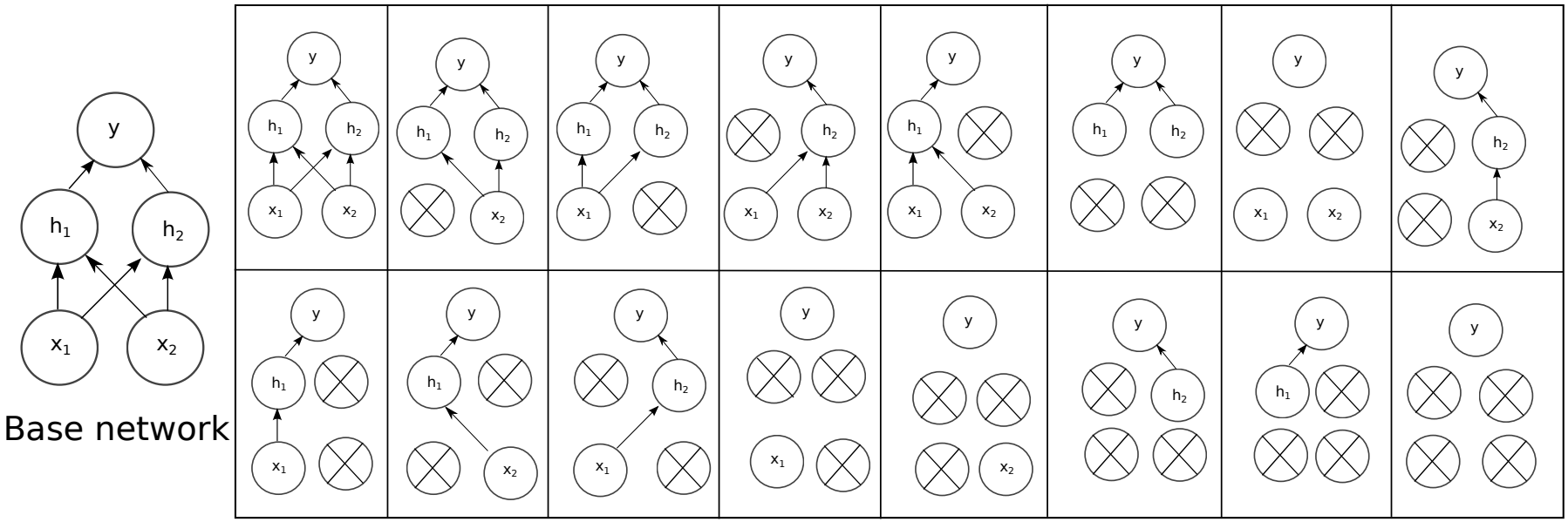
- all probabilistic predictions are (geometrically) averaged
  - the distribution factorizes, i.e. $P(\underline{\mathbf{d}}) = \prod_i P(d_i)$

$$P(y|\underline{\mathbf{x}}) \quad = \quad \sqrt[2^n]{\prod_{\underline{\mathbf{d}} \in \{0,1\}^n} P(\underline{\mathbf{d}})\, P\big(y \,|\, \underline{\mathbf{d}} \cdot \underline{\mathbf{S}}(\underline{\mathbf{x}})\big)} \quad \approx \quad P\big(y \,|\, \mathbb{E}[\underline{\mathbf{d}}] \cdot \underline{\mathbf{S}}(\underline{\mathbf{x}})\big)$$

- approximation can rarely be proven, but works well empirically

(e.g. the approximation is exact for a softmax layer, Goodfellow et al., 2016)

Dropout as an ensemble method



Base network

- prediction uses all neurons, but scales the learned weights

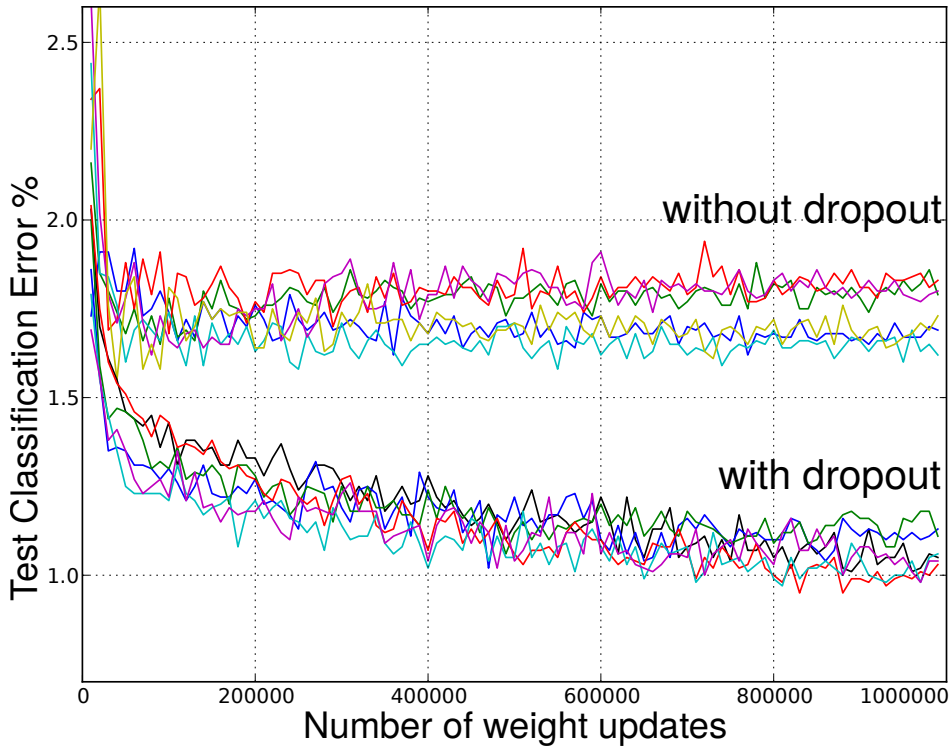
$$w_{ij}^{v'v} \quad \leftarrow \quad w_{ij}^{v'v} \cdot \mathbb{E}[d_j^v]$$

- the prediction approximates the geometric ensemble average

(see Goodfellow et al., 2016)

Dropout regularizes deep networks

- solving MNIST classification with different architectures
  - sigmoid MLPs with 1024 or 2048 neurons in each of 2, 3 or 4 layers
  - neurons were sampled $\mathbb{E}[d_i] = 0.5$ and input variables $\mathbb{E}[d_j] = 0.8$

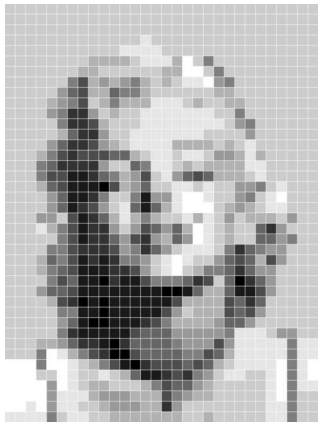


(results from Srivastava et al., 2014)

1.5.4 Convolutional Layers

Structured input and output

$1 \times n$
tensor



$n \times m$
tensor

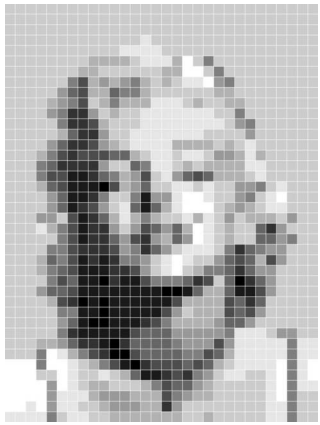
$n \times m \times 3$
tensor

$n \times m \times 3 \times t$
tensor

- structured input/output spaces are sometimes called **tensors**
  - here defined as multidimensional extensions of matrices
  - tensors imply a *spatial* relationship between variables

Structured input and output

$1 \times n$ tensor

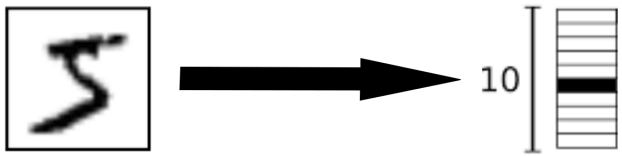


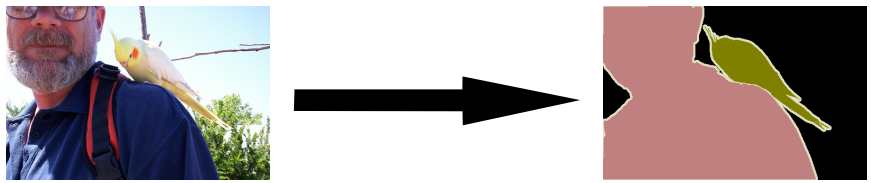
$n \times m$ tensor

$n \times m \times 3$ tensor

$n \times m \times 3 \times t$ tensor

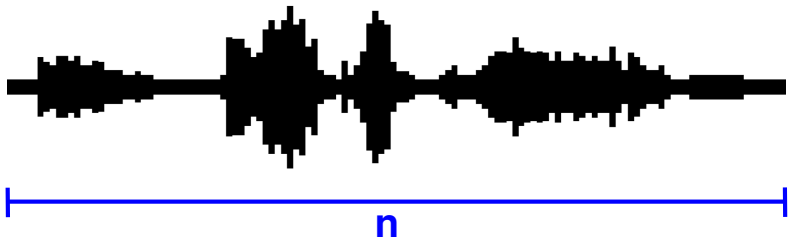
- MLP can have tensors as inputs and as outputs, e.g.
  - MNIST has a 28×28 input tensor and a 10×1 output vector
  - segmentation to predict the location of c object-classes in an image has a $n \times m \times 3$ input tensor and a $n \times m \times c$ output tensor





Convolutions

- convolutions require a spatial structure, i.e. an input tensor
- Example: one-dimensional audio data
  - $\underline{\mathbf{x}}$ is a $1 \times n$ input tensor
  - $\underline{\mathbf{w}}$ is a $1 \times n'$ filter tensor
  - $\underline{\mathbf{h}}$ is a $1 \times (n-n')$ output tensor



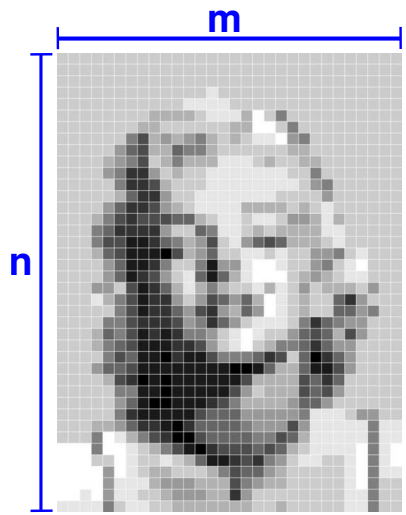
$$h_i = (\underline{\mathbf{x}} * \underline{\mathbf{w}})_{(i)} = \sum_{k=1}^{n-n'} x_k \, w_{i-k} = \sum_{k=1}^{n'} x_{i-k} \, w_k \qquad \text{(convolution)}$$

$$h_i = (\underline{\mathbf{x}} \star \underline{\mathbf{w}})_{(i)} = \sum_{k=1}^{n-n'} x_k \, w_{i+k} = \sum_{k=1}^{n'} x_{i+k} \, w_k \qquad \text{(correlation)}$$

Convolutions

- convolutions require a spatial structure, i.e. an input tensor

- Example: two-dimensional gray-scale images
  - $\underline{\mathbf{X}}$ is a $n \times m$ input tensor
  - $\underline{\mathbf{W}}$ is a $n' \times m'$ filter tensor
  - $\underline{\mathbf{H}}$ is a $(n-n') \times (m-m')$ output tensor

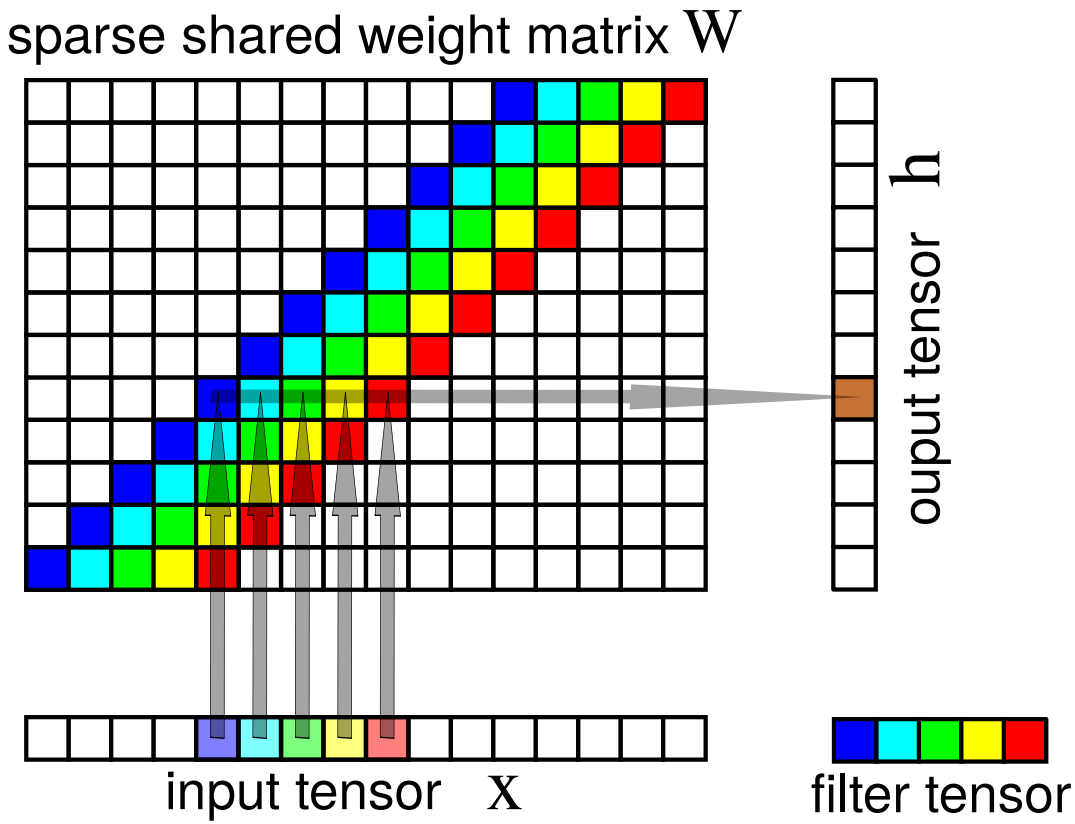


$$\underline{\mathbf{H}}_{(i,j)} = (\underline{\mathbf{X}} * \underline{\mathbf{W}})_{(i,j)} = \sum_{k=1}^{n} \sum_{l=1}^{m} \underline{\mathbf{X}}_{(k,l)} \, \underline{\mathbf{W}}_{(i-k,j-l)}$$

$$= \sum_{k=1}^{n'} \sum_{l=1}^{m'} \underline{\mathbf{X}}_{(i-k,j-l)} \, \underline{\mathbf{W}}_{(k,l)}$$

Convolutions reduce number of parameters

- convolutions correspond to a sparsely connected shared weight matrix



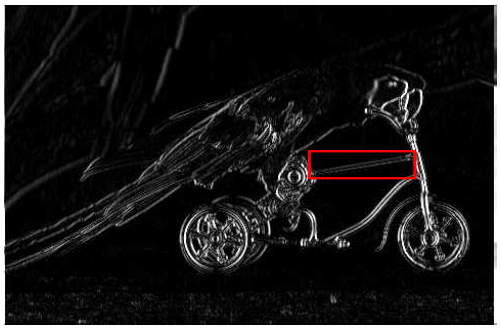
- sparse shared weight matrix $\underline{\mathbf{W}} \in \mathbb{R}^{n \times n-n'}$ has only n' parameters
- shifted filters enforce a location-invariant feature

Feature maps

- most tasks require multiple features, e.g. horizontal and vertical edges

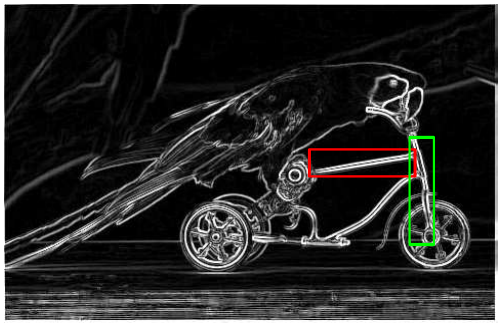
$\underline{\mathbf{X}}$

$\underline{\mathbf{S}}_1(\underline{\mathbf{X}}) = |\underline{\mathbf{X}} * \underline{\mathbf{W}}_1|$



$$\underline{\mathbf{W}}_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

"vertical edges" feature



$\underline{\mathbf{S}}_1 + \underline{\mathbf{S}}_2$

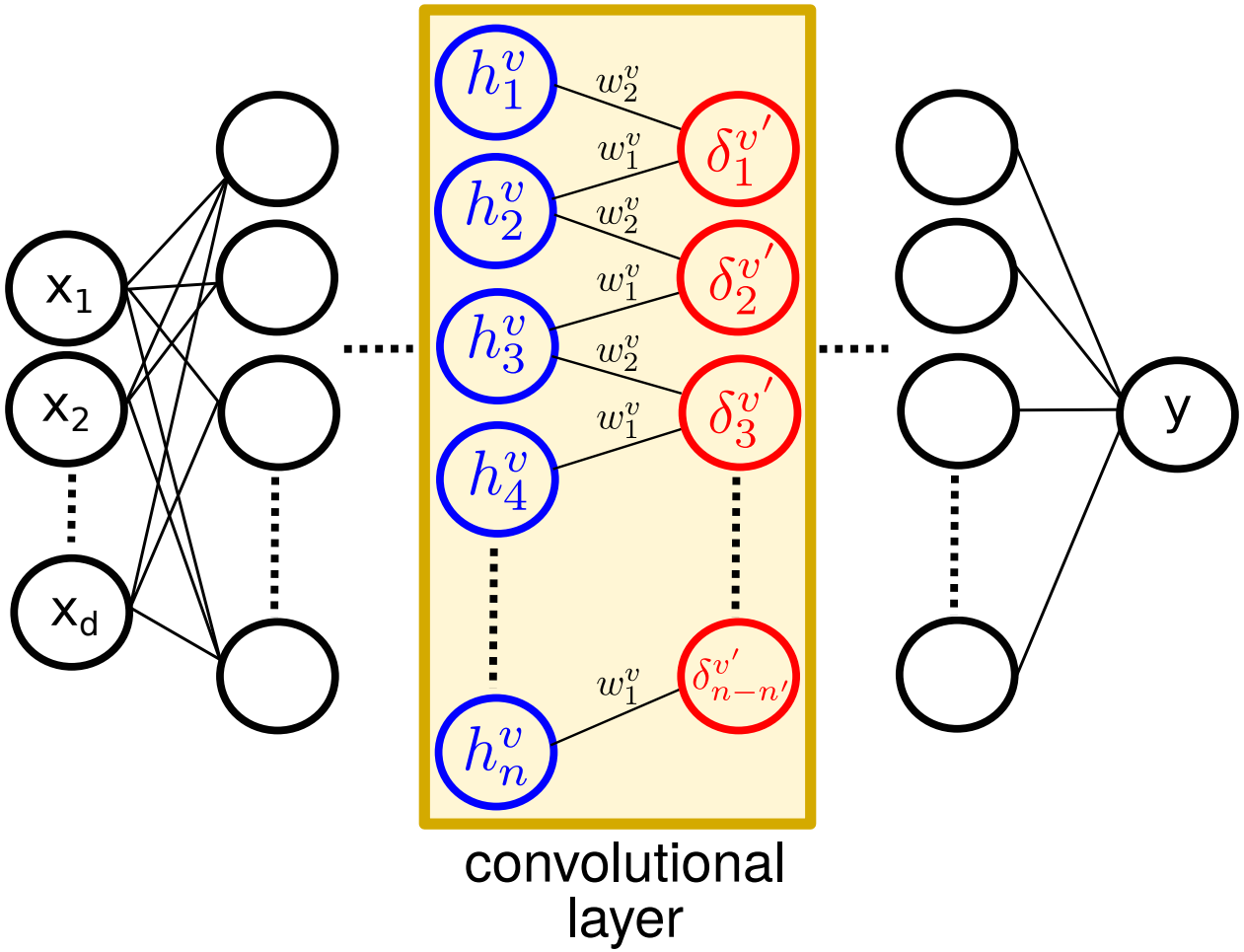
$\underline{\mathbf{S}}_2(\underline{\mathbf{X}}) = |\underline{\mathbf{X}} * \underline{\mathbf{W}}_2|$

$$\underline{\mathbf{W}}_2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

"horizontal edges" feature

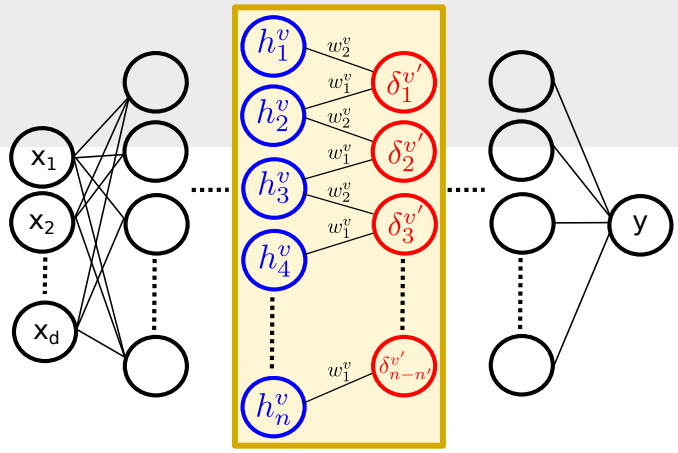
- $n \times m \times 3$ input tensor $\longrightarrow$ $(n-3) \times (m-3) \times 2$ output tensor

Convolutional layers in MLP



- convolutional layer gets activities h_i^v from the preceding layer
- convolutional layer gets errors $\delta_j^{v'}$ from the succeeding layer

Training convolutional layers



1. gradient calculation by chain rule:

$$\frac{\partial e^{(\alpha)}}{\partial w_k^v} = \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot \frac{\partial h_j^{v'}}{\partial w_k^v} = \Big[\overbrace{\frac{\partial e^{(\alpha)}}{\partial \underline{\mathbf{h}}^{v'}}}^{\text{error } \underline{\delta}^{v'}} \star \underbrace{f\big(\underline{\mathbf{h}}^v\big)}_{\text{activity } \underline{\mathbf{h}}^v} \Big]_{(n'-k)}$$

2. forward pass:

$$h_j^{v'} := \big[f(\underline{\mathbf{h}}^v) * \underline{\mathbf{w}}^v\big]_{(j)} = \sum_{k=1}^{n'} f\big(h_{n'+j-k}^v\big) \cdot w_k^v\,, \qquad \forall j \in \{1,\ldots,n-n'\}$$

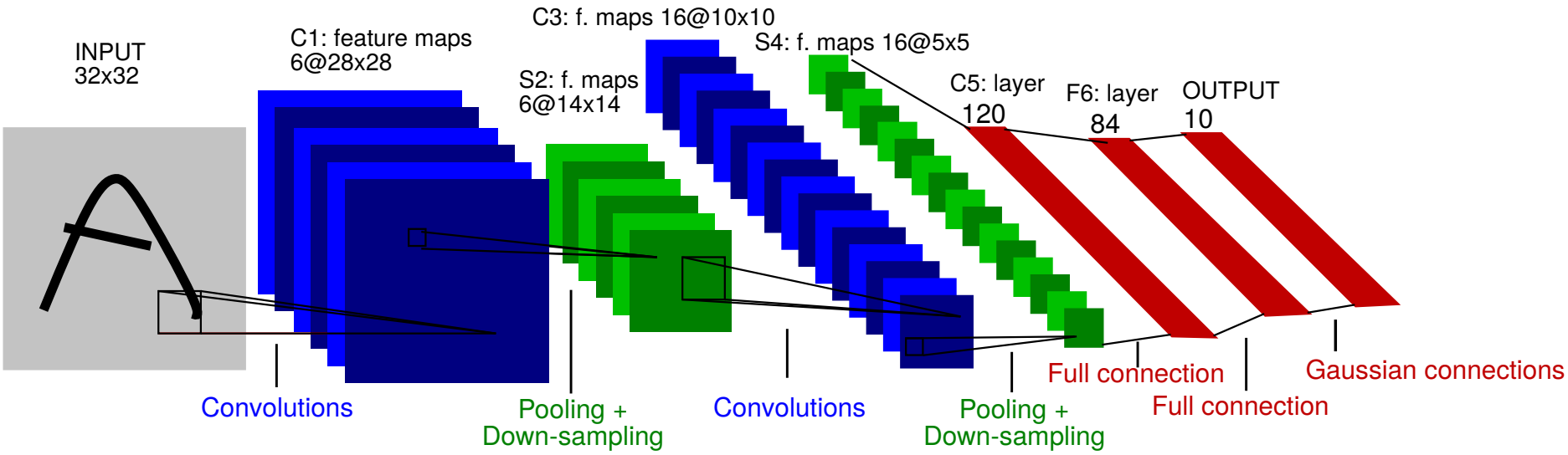
3. backward pass:

$$\delta_i^v = f'\big(h_i^v\big) \cdot \big[\underline{\delta}^{v'} \star \underline{\mathbf{w}}^v\big]_{(i-n')} = f'\big(h_i^v\big) \cdot \sum_{j=1}^{n'} \delta_{(i-n')+j}^{v'} \cdot w_j^v$$

derivation here

1.5.5 Convolutional Architectures

Common deep learning architectures



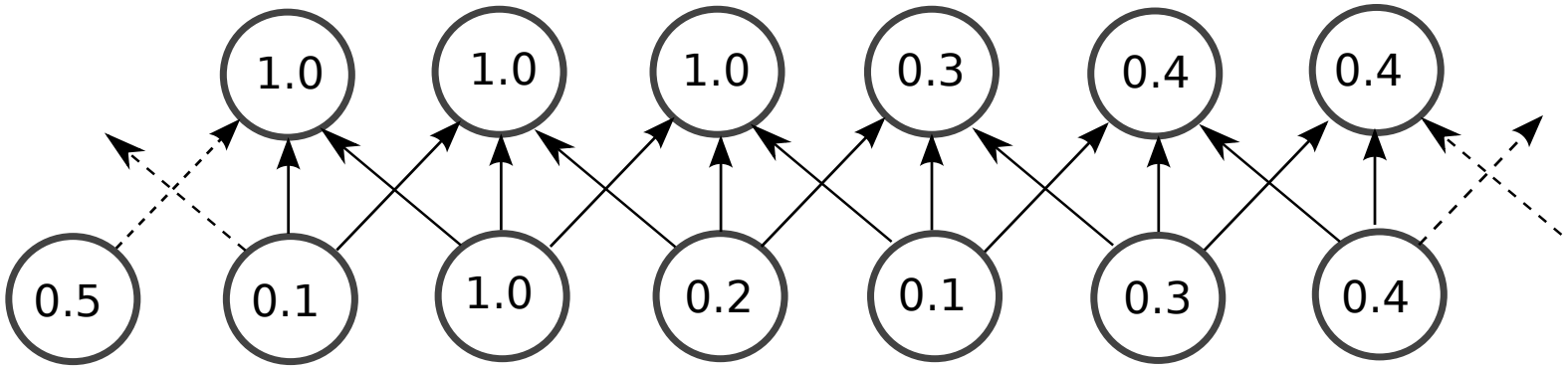
- most modern deep architecture consist of:
  - a larger number of convolutional layers (increase #features)
  - pooling and down-sampling layers (decrease size)
  - a smaller number of fully connected layers

(LeNet-5 architecture to recognize 10 numbers, LeCun et al., 1998)

Introducing invariances: spatial pooling

- invariance to small spatial transformation, e.g.

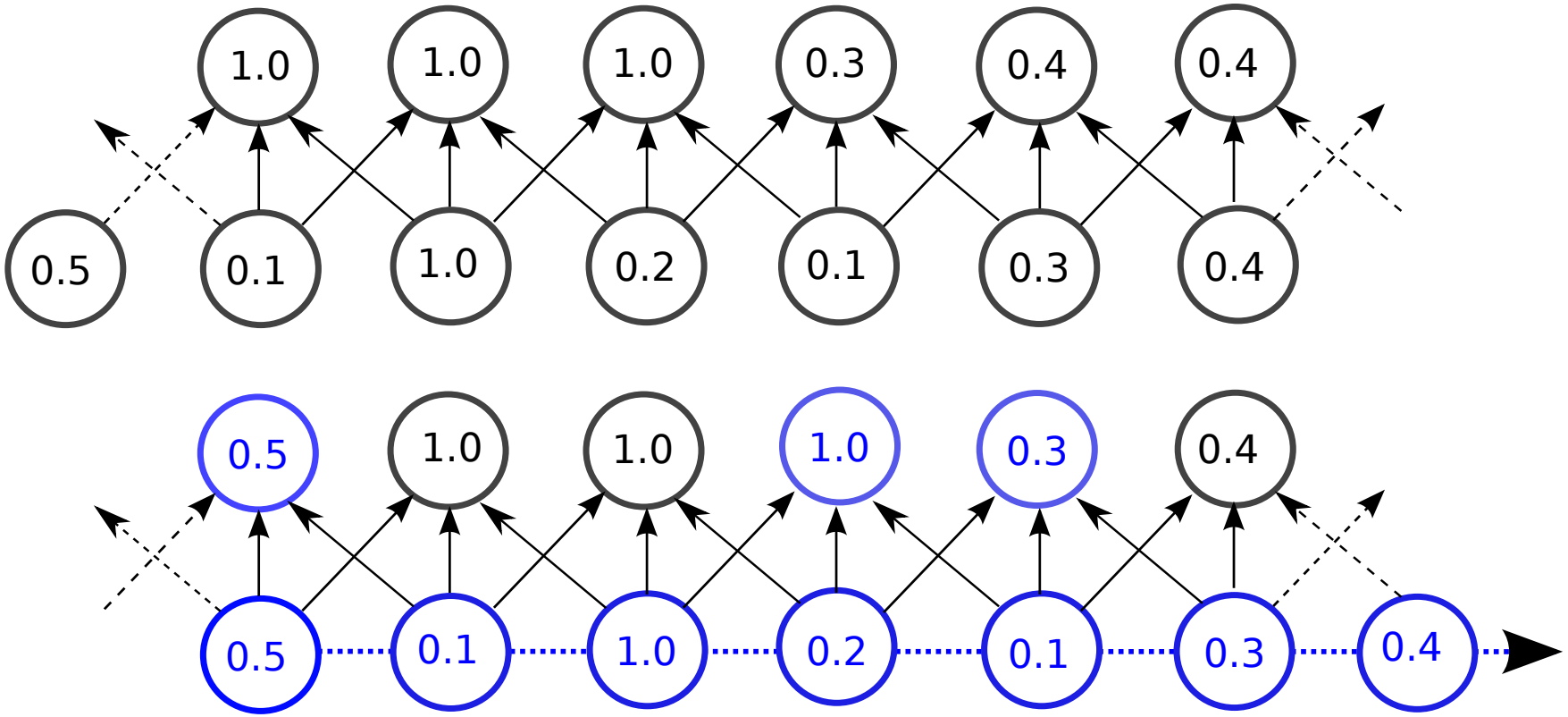
$$f_i(\underline{\mathbf{x}}) = \max(x_{i-m}, x_{i-m+1}, \ldots, x_{i+m-1}, x_{i+m})$$



Introducing invariances: spatial pooling

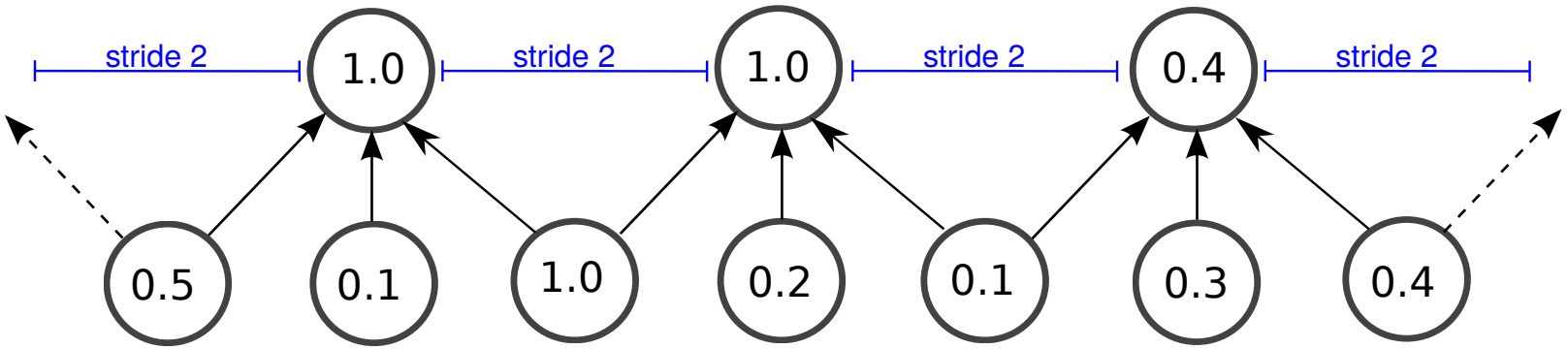
- invariance to small spatial transformation, e.g.

$$f_i(\underline{\mathbf{x}}) = \max(x_{i-m}, x_{i-m+1}, \ldots, x_{i+m-1}, x_{i+m})$$



Down-sampling

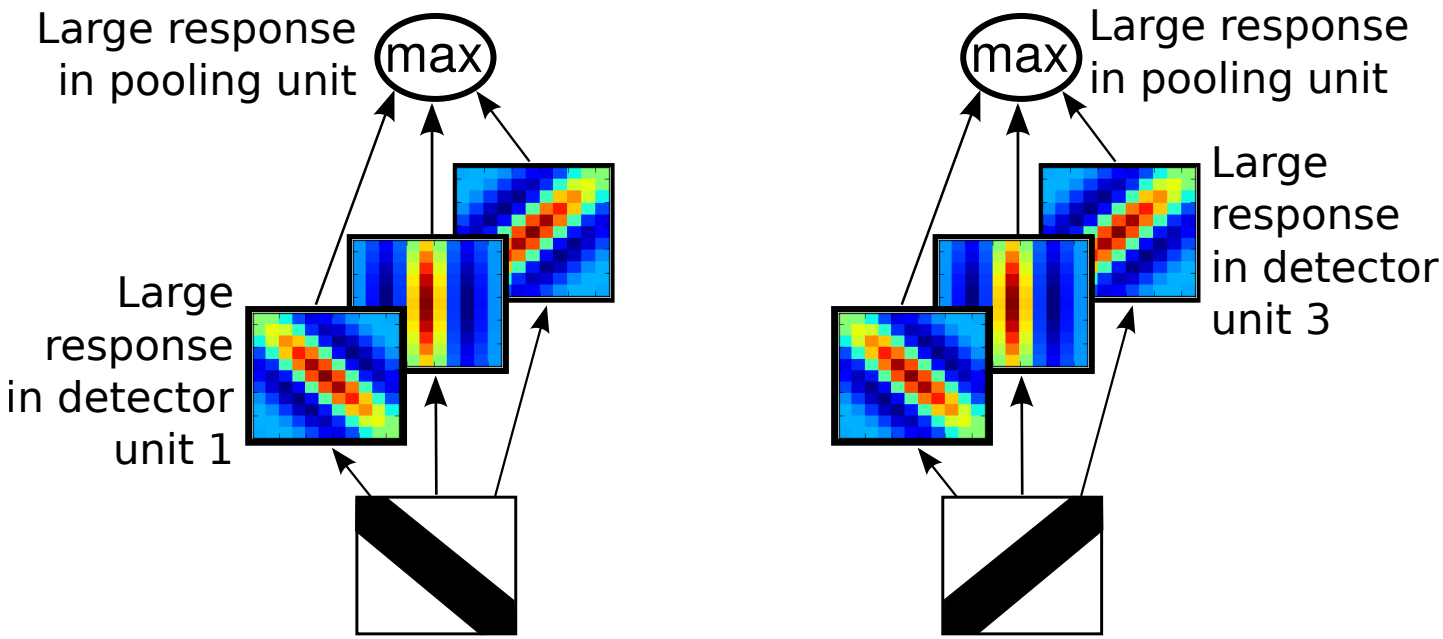
- spatial invariance (from pooling) can be exploited by down-sampling
  - using only every *stride*'th output node, e.g. for stride 2:



- down-sampling yields more compact representation
  - smaller output tensors $\rightsquigarrow$ less parameters in successive layers

Introducing invariances: feature pooling

- other invariances can be learned by pooling over multiple features
  - each filter learns to detect the feature in another pose
  - the pooling layer keeps the maximum of the participating filters
  - the pooled output is invariant against the learned transformation

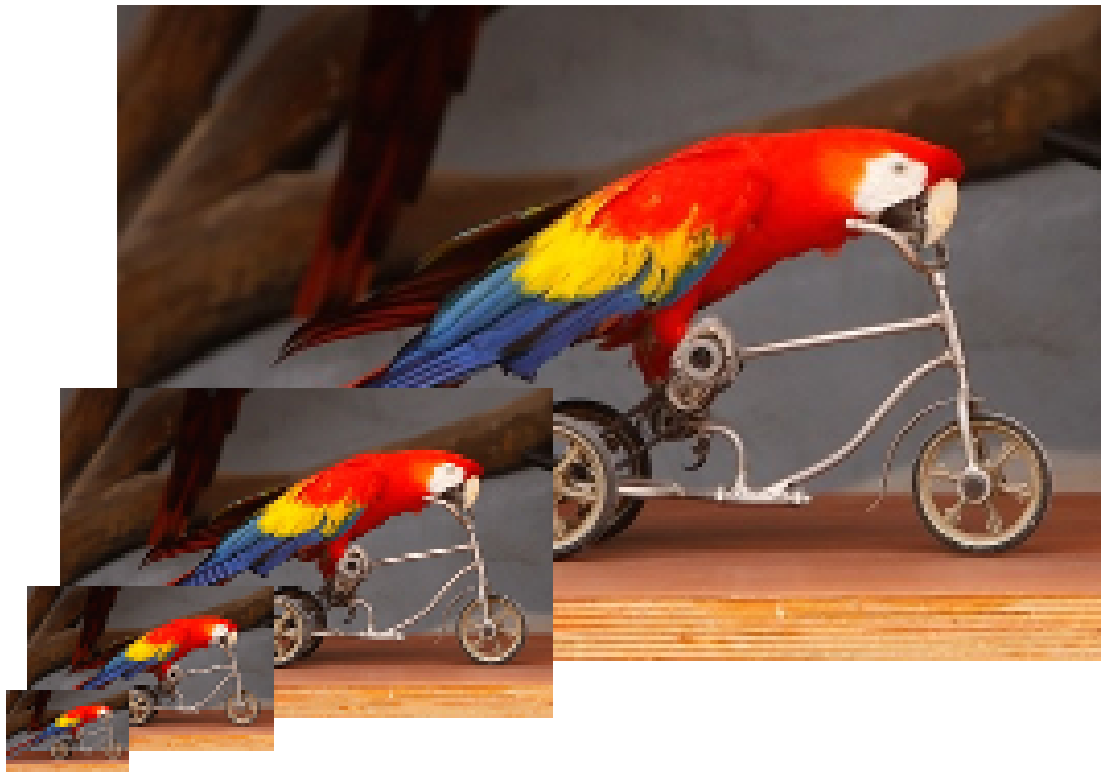


- training determines which invariances are learned

(maxout networks, Goodfellow et al., 2013)

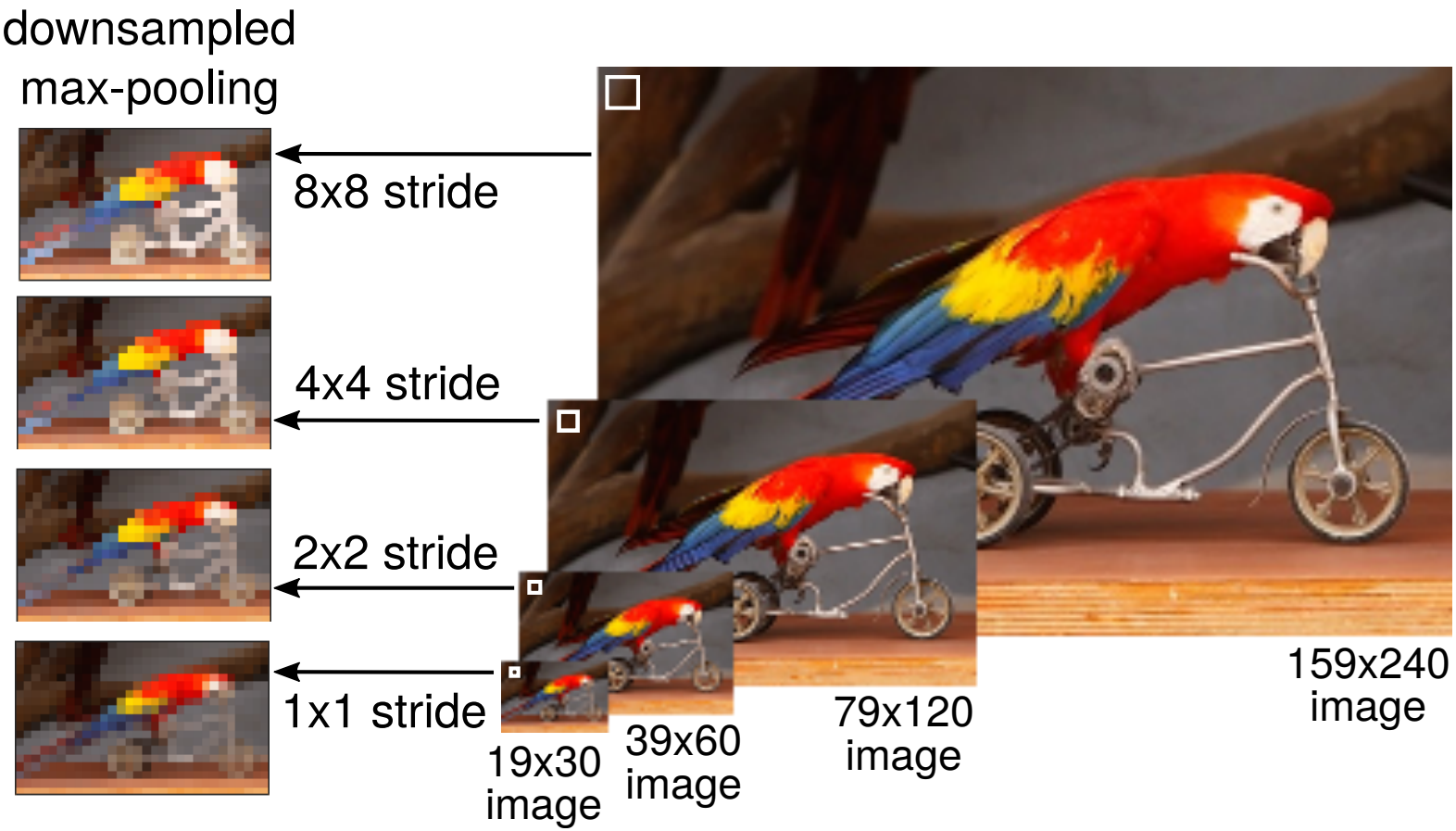
Input tensors with varying size

- images come in many sizes and resolutions
  - but fully connected layers have a fixed input size
  - convolution is size-free and downsampling strides are flexible

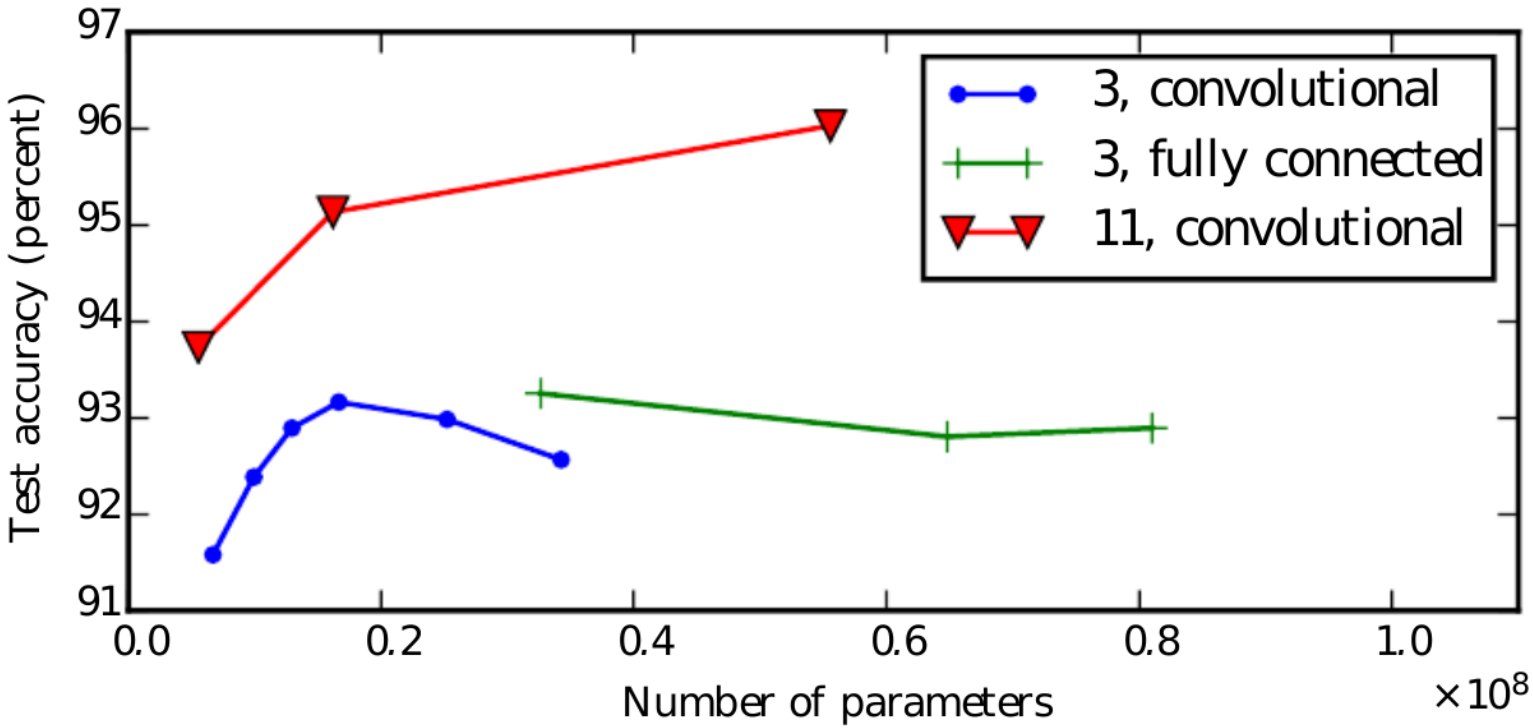


Input tensors with varying size

- one can exploit *downsampling* to reduce the image size
  - choose *stride* in each layer such that the input size of the first fully connected layer is constant



Depth and parameter sizes

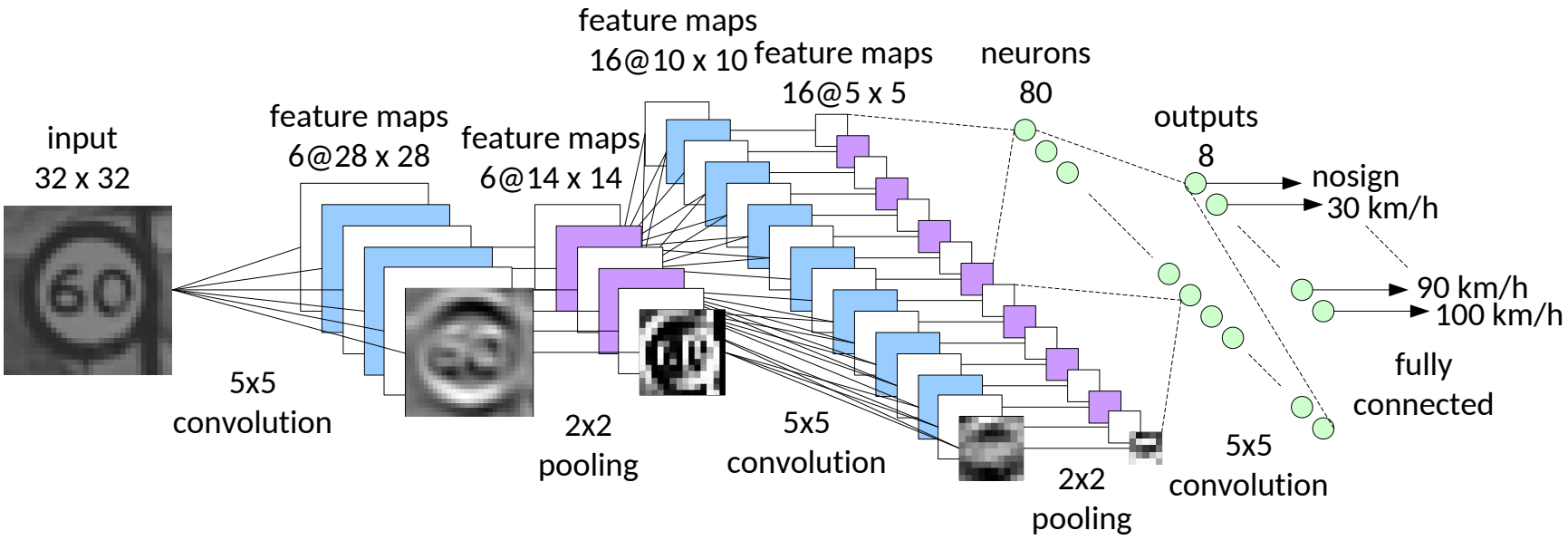


- convolutions *reduce parameters*, which allows for more layers
- convolutions *regularize* by restricting the model class
- convolutions *generalize* better due to introduced invariances

(house-number recognition on street-view images, Goodfellow et al., 2014)

Example deep architecture: detecting traffic signs

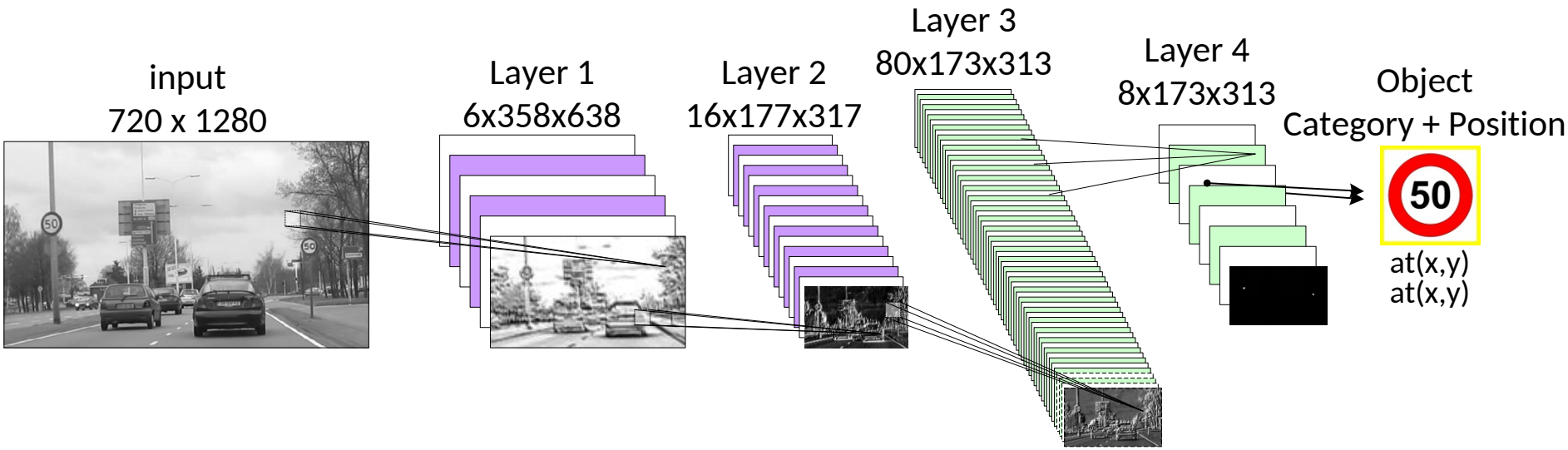
- detection of speed related traffic signs for a fixed input size
  - small 32×32 input images of traffic signs
  - three 5×5 convolution layers with 2×2 downsampled max-pooling
  - each convolution layer increases the features, but reduces the size
  - one fully connected layer predicts the traffic sign



(figure and architecture from Peemen et al., 2016)

Example deep architecture: detecting traffic signs

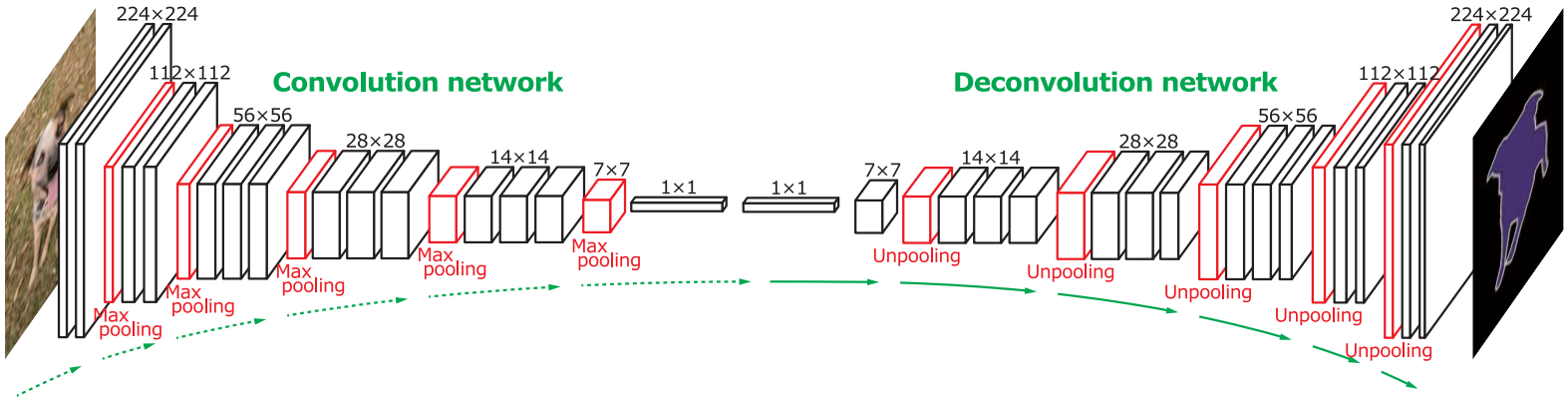
- simultaneous detection of speed limits and traffic sign position
  - apply sign-recognition network on every *tile* of a 720×1280 image
  - each application predicts which traffic sign is present in the tile
  - final map detects all visible traffic signs and their position



- network runs real-time on customized hardware (e.g. in a car)

(figure and architecture from Peemen et al., 2016)

Deconvolutional networks

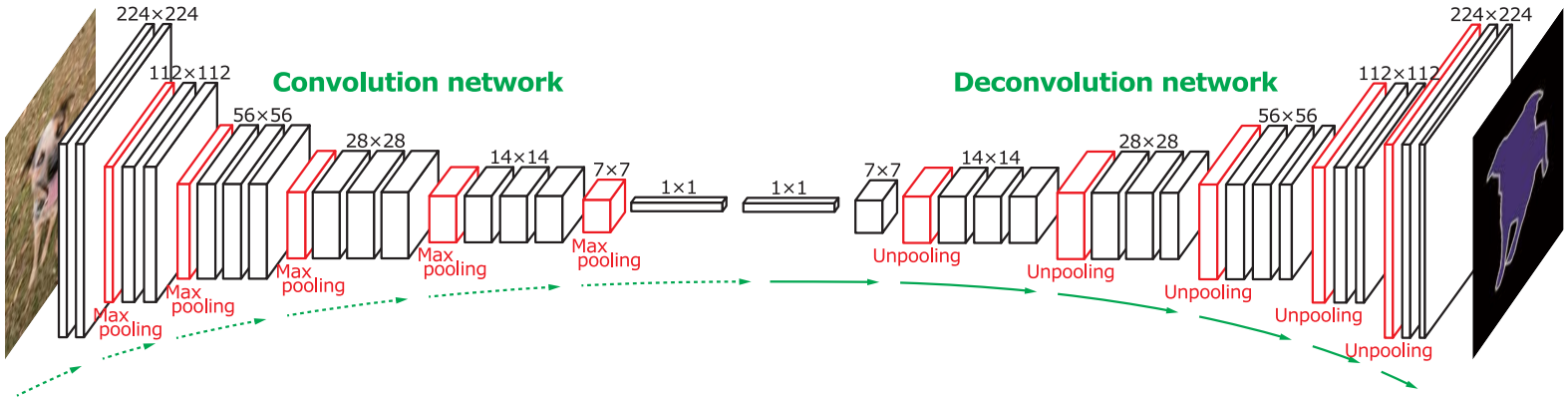


- MLP output tensors can be equal (or larger) than the input tensor
  - e.g. image segmentation: identify which pixel belongs to which object

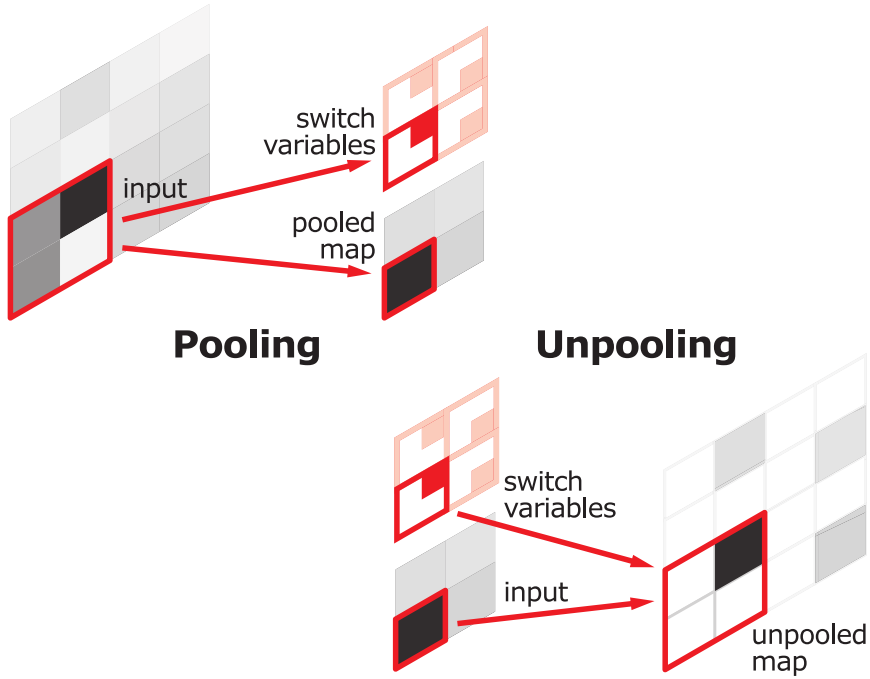
- original convolution and pooling layers must be *reversed*
  - deconvolutional (sub-)networks have larger output tensors

(figures and architecture from Noh et al., 2015)

Unpooling layers

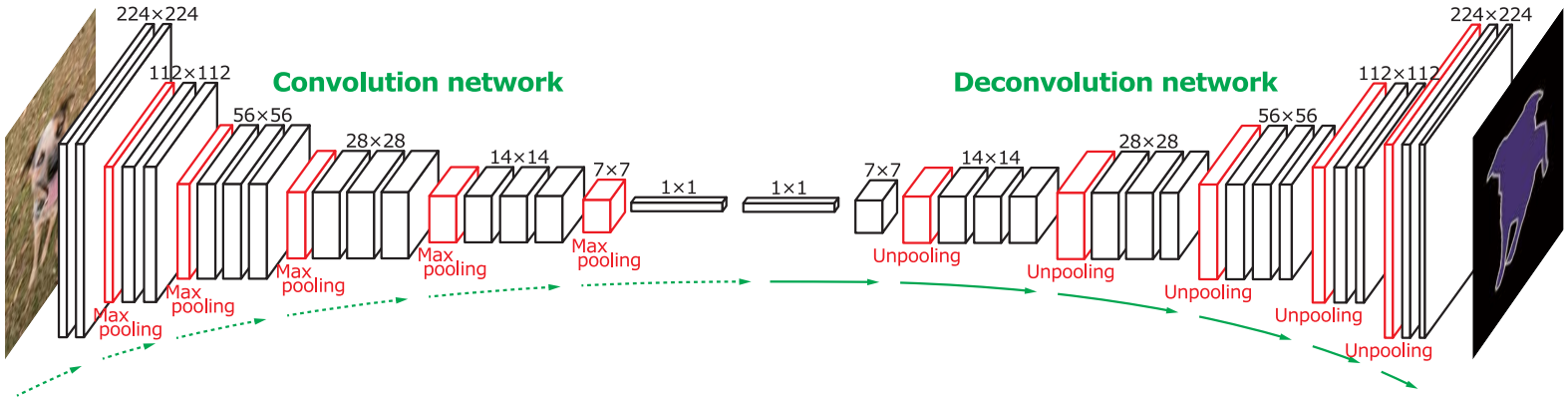


- every pooling layer can be reversed by an unpooling layer
- remember "switch-variables", i.e. which unit was maximal
- yields enlarged, but sparse feature maps

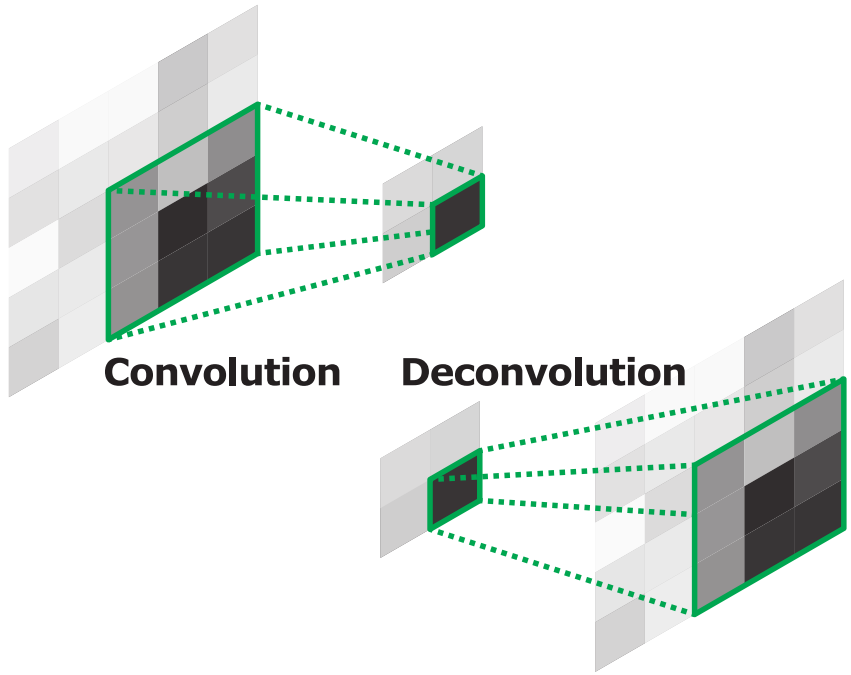


(figures and architecture from Noh et al., 2015)

Deconvolutional layers

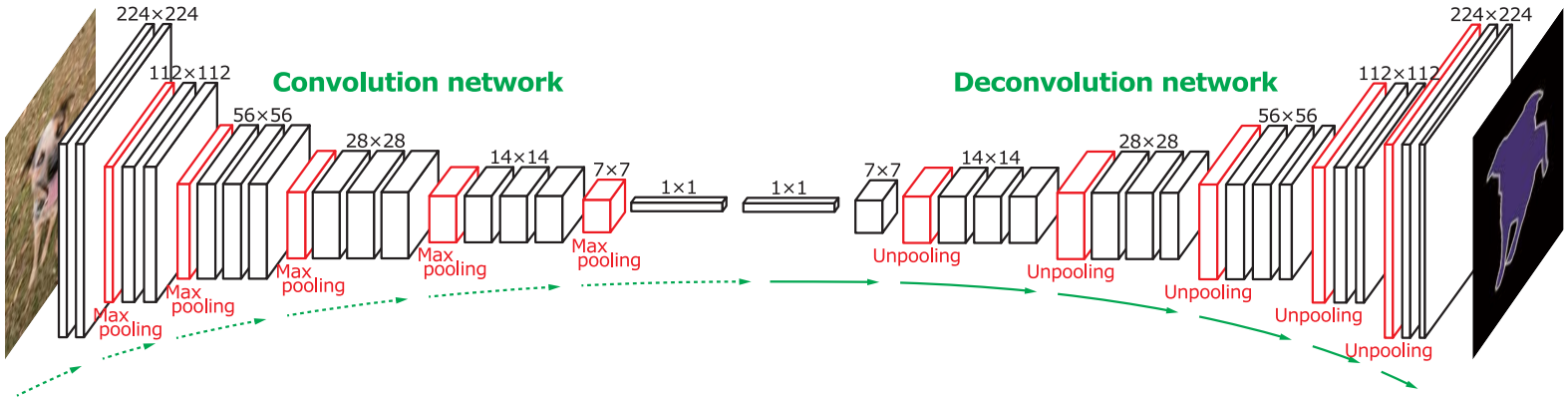


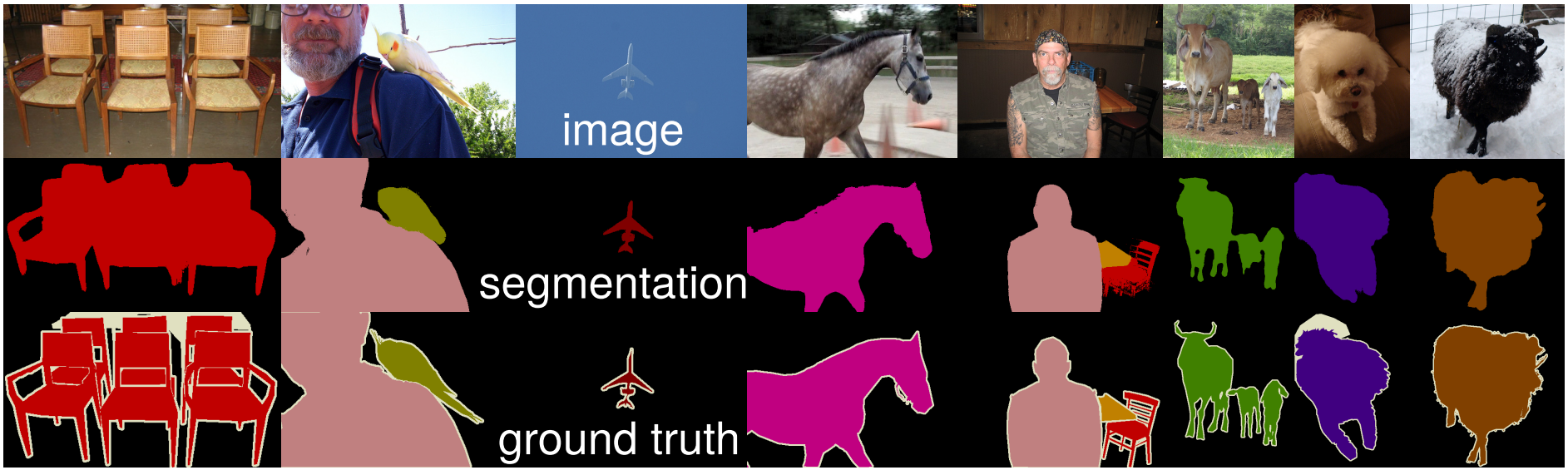
- every convolution layer can be reversed by a deconvolution layer
- learns small input tensors to predict large output tensors
- convolution generates dense feature maps from sparse unpooling maps



(figures and architecture from Noh et al., 2015)

Deconvolutional example: image segmentation





(trained on PASCAL VOC 2012 benchmark data, Noh et al., 2015)

End of Section 1.5

the following slides contain

OPTIONAL MATERIAL

Training convolutional layers: derivation

- forward pass:

$$h_j^{v'} := [f(\underline{\mathbf{h}}^v) * \underline{\mathbf{w}}^v]_{(j)} = \sum_{k=1}^{n'} f\left(h^v_{n'+j-k}\right) \cdot w_k^v\,, \qquad \forall j \in \{1, \ldots, n-n'\}$$

Training convolutional layers: derivation

- forward pass:

$$h_j^{v'} := [f(\underline{\mathbf{h}}^v) * \underline{\mathbf{w}}^v]_{(j)} = \sum_{k=1}^{n'} f\left(h^v_{n'+j-k}\right) \cdot w_k^v\,, \qquad \forall j \in \{1,\ldots,n-n'\}$$

- gradient calculation:

$$\begin{aligned}
\frac{\partial e^{(\alpha)}}{\partial w_k^v} &= \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot \frac{\partial h_j^{v'}}{\partial w_k^v} \\
&= \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot f\left(h^v_{n'+j-k}\right) \\
&= \Bigg[\underbrace{\frac{\partial e^{(\alpha)}}{\partial \underline{\mathbf{h}}^{v'}}}_{\underline{\delta}^{v'}} \star \underbrace{f\left(\underline{\mathbf{h}}^v\right)}_{\text{input}} \Bigg]_{(n'-k)}
\end{aligned}$$

Training convolutional layers: derivation

- forward pass:

$$h_j^{v'} := [f(\underline{\mathbf{h}}^v) * \underline{\mathbf{w}}^v]_{(j)} = \sum_{k=1}^{n'} f\left(h^v_{n'+j-k}\right) \cdot w^v_k\,, \qquad \forall j \in \{1,\ldots,n-n'\}$$

- gradient calculation:

$$\begin{aligned}
\frac{\partial e^{(\alpha)}}{\partial w_k^v} &= \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot \frac{\partial h_j^{v'}}{\partial w_k^v} \\
&= \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot f\left(h^v_{n'+j-k}\right) \\
&= \Bigg[\underbrace{\frac{\partial e^{(\alpha)}}{\partial \underline{\mathbf{h}}^{v'}}}_{\underline{\delta}^{v'}} \star \underbrace{f\left(\underline{\mathbf{h}}^v\right)}_{\text{input}} \Bigg]_{(n'-k)}
\end{aligned}$$

- backward pass:

$$\begin{aligned}
\delta_i^v &= \sum_{j=1}^{n-n'} \frac{\partial e^{(\alpha)}}{\partial h_j^{v'}} \cdot \frac{\partial h_j^{v'}}{\partial f(h_i^v)} \cdot \frac{\partial f(h_i^v)}{\partial h_i^v} \\
&= \sum_{j=1}^{n-n'} \delta_j^{v'} \cdot w^v_{n'+j-i} \cdot f'\left(h_i^v\right) \\
&= \sum_{j=1}^{n'} \delta^{v'}_{i-n'+j} \cdot w_j^v \cdot f'\left(h_i^v\right) \\
&= \left[\underline{\delta}^{v'} \star \underline{\mathbf{w}}^v\right]_{(i-n')} \cdot f'\left(h_i^v\right)
\end{aligned}$$

- note that gradient and backward-pass are computed with a correlation

References I

Yoshua Bengio. Learning deep architectures for ai. *Foundations Trends in Machine Learning*, 2 (1):1–127, 2009.

Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe. Phonetically motivated acoustic parameters for continuous speech recognition using artificial neural networks. *Speech Communication*, 11(2):261–271, 1992. Eurospeech '91.

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *International Conference of Learning Representation (ICLR)*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, David Warde-farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Internation Conference on Machine Learning (ICML)*, 2013.

G. E. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

References II

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2924–2932, 2014.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, 2015.

Maurice Peemen, Runbin Shi, Sohan Lal, Ben Juurlink, Bart Mesman, and Henk Corporaal. The Neuro Vector Engine: Flexibility to improve convolutional net efficiency for wearable vision. In *Design, Automation and Test in Europe (DATE)*, 2016.

R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 873–880, 2009.

F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, 1962.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014.

References III

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *13th European Conference of Computer Vision (ECCV)*, pages 818–833, 2014.