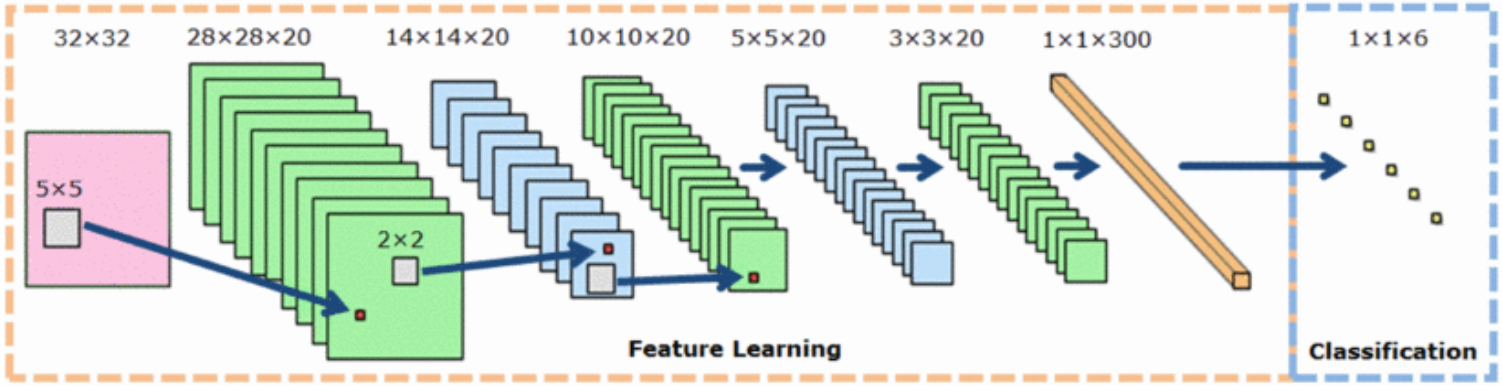
Going Deeper with
(Convolutional) Neural Networks



M.Sc. Oskar Maier

Institut für Medizinische Informatik

Universität zu Lübeck

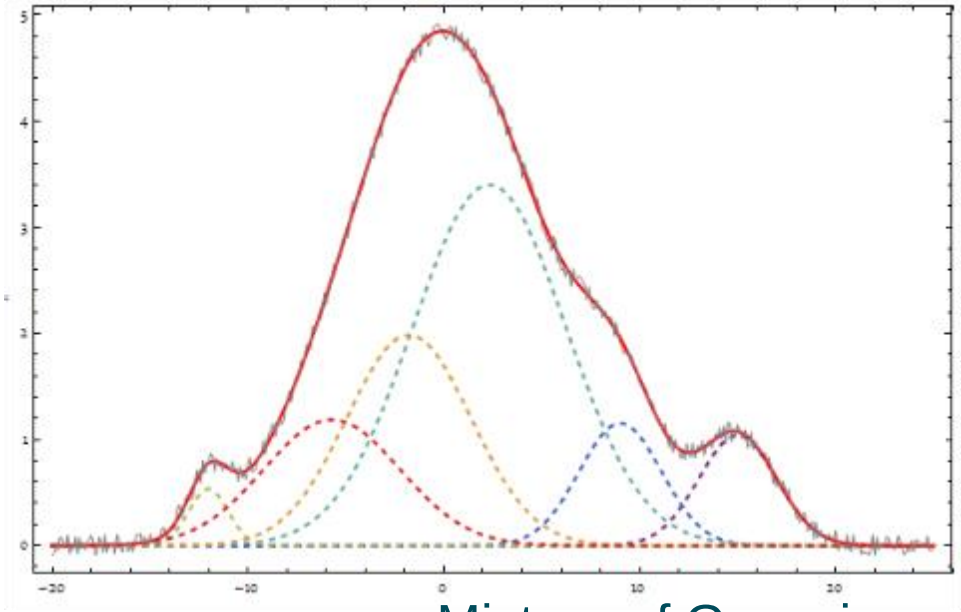
Deep Neural Networks

Theory

Two layers are sufficient to model any functions

But

These networks do not learn well in practice



Mixture of Gaussians

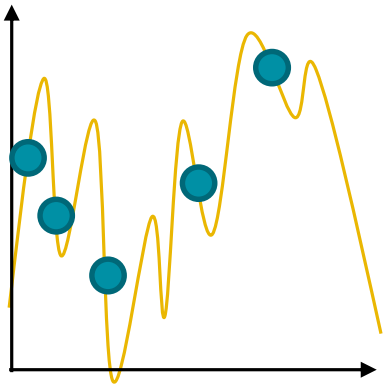
Alternative: Going deeper with Deep Neural Networks (DNN)

- Multiple layers allow for the modelling of the most complex function with (comparably) less parameters
- Small concepts are joined gradually into larger ones

Image source: http://i.stack.imgur.com/tJXqK.png

Problems with deep networks

1. Amount of parameters
2. Overfitting
3. Gradient vanishing/explosion
4. Input vanishing/explosion

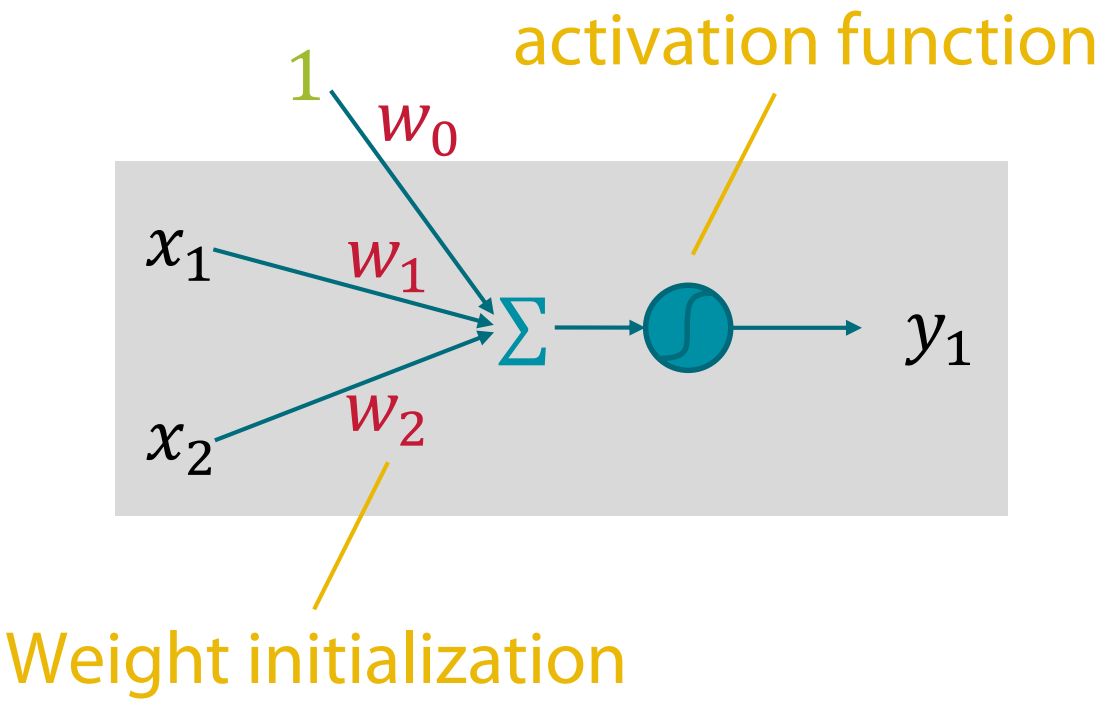


$$w < 1$$
$$w > 1$$

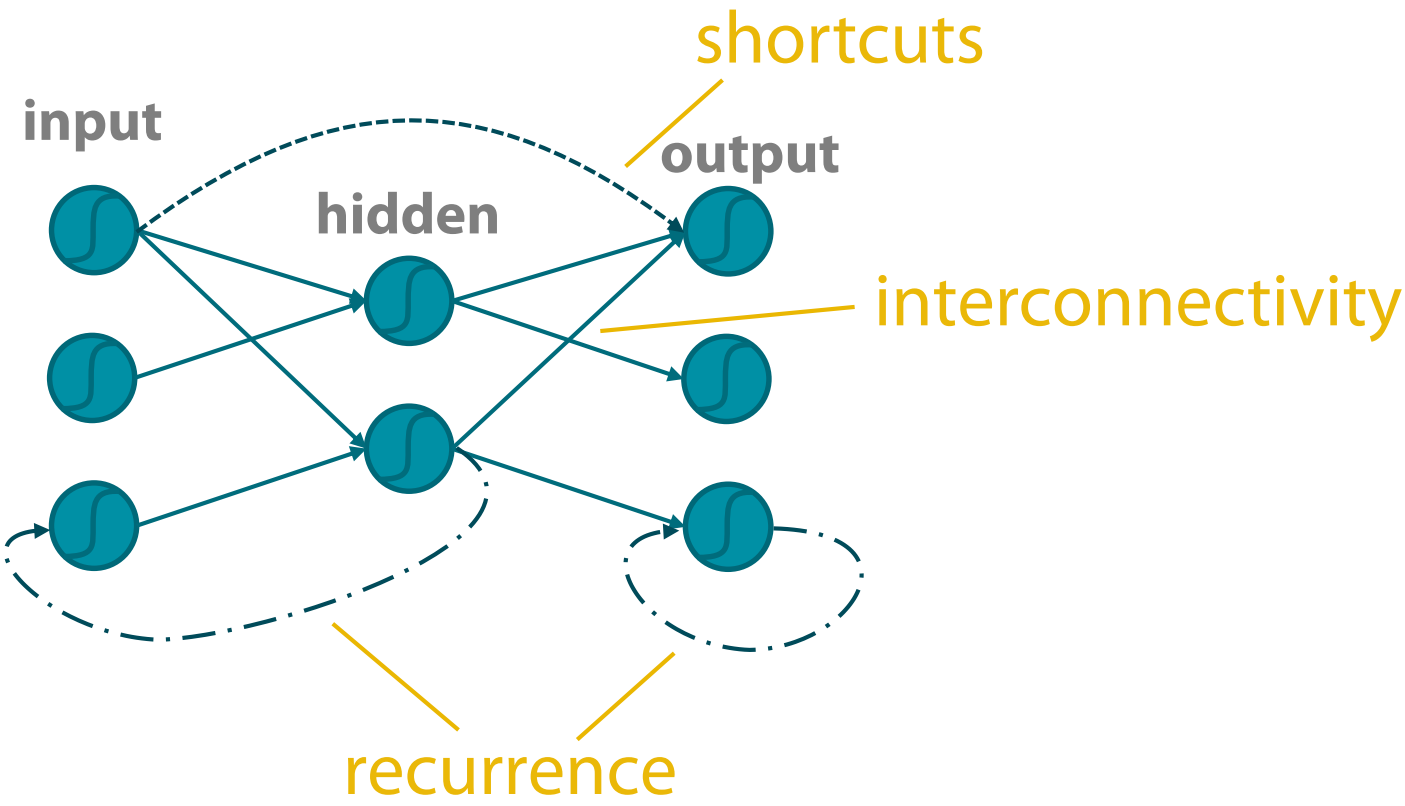
Summary: Many problems

- Some of them application dependent

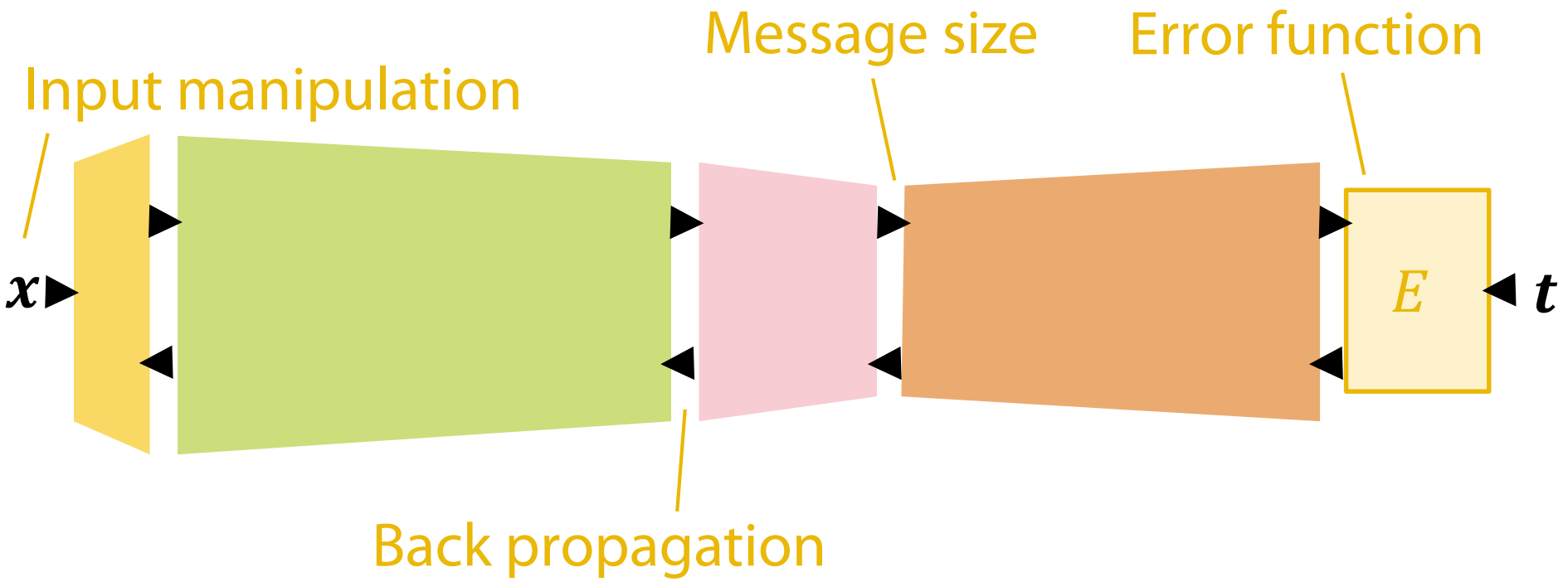
Leverage points



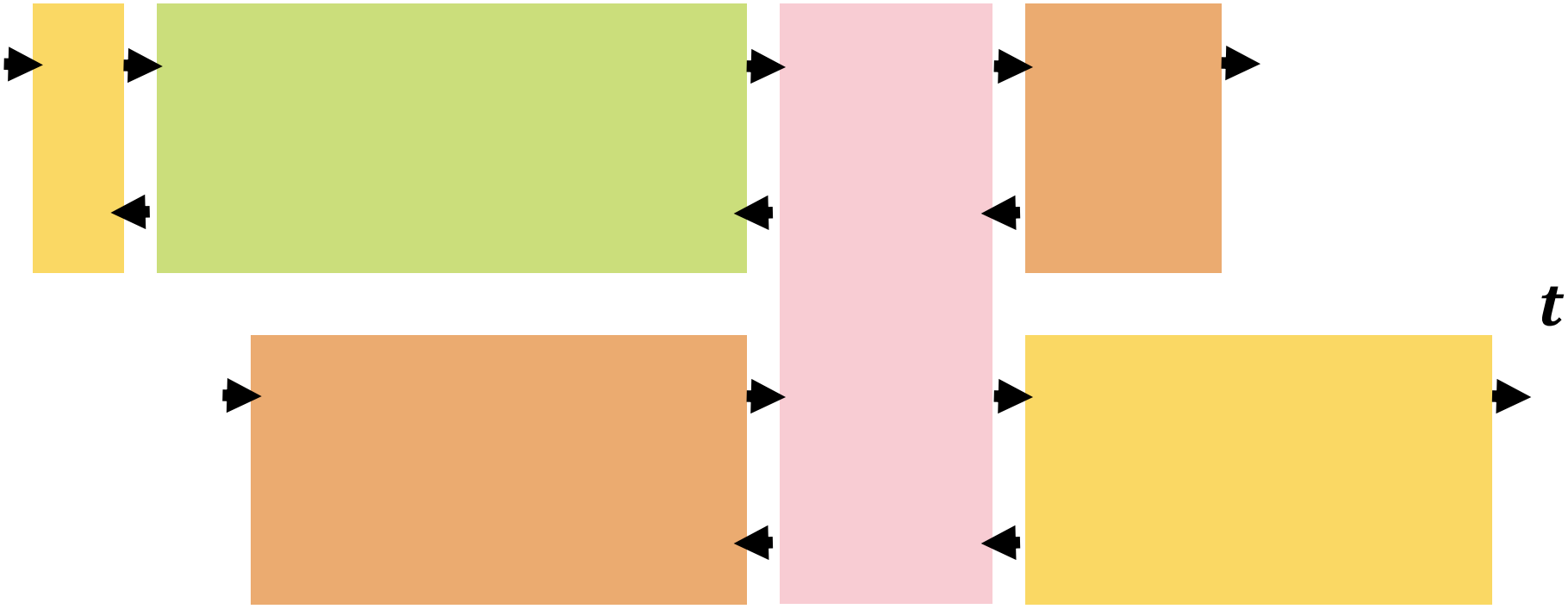
Leverage points



Leverage points



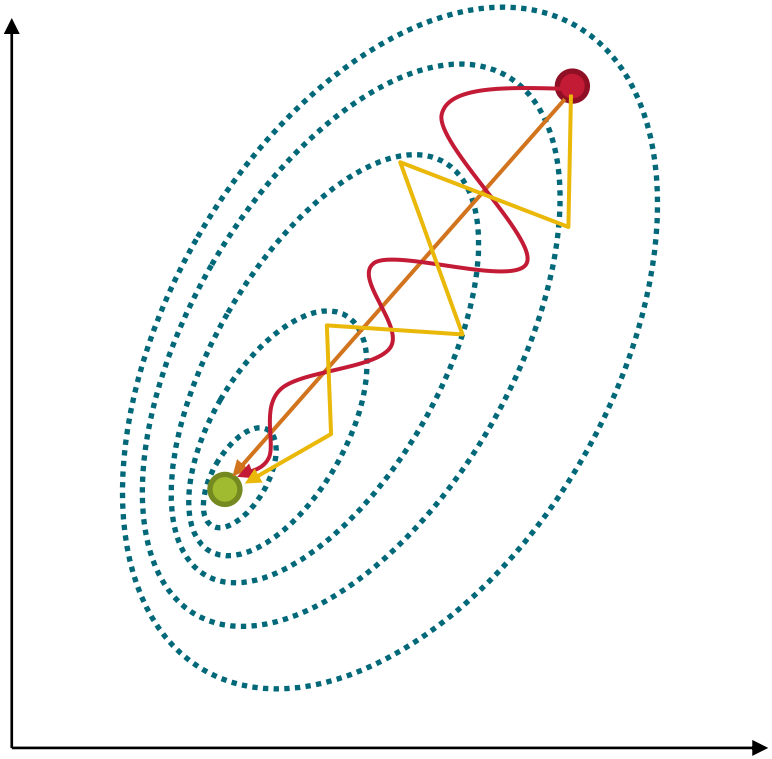
Leverage points



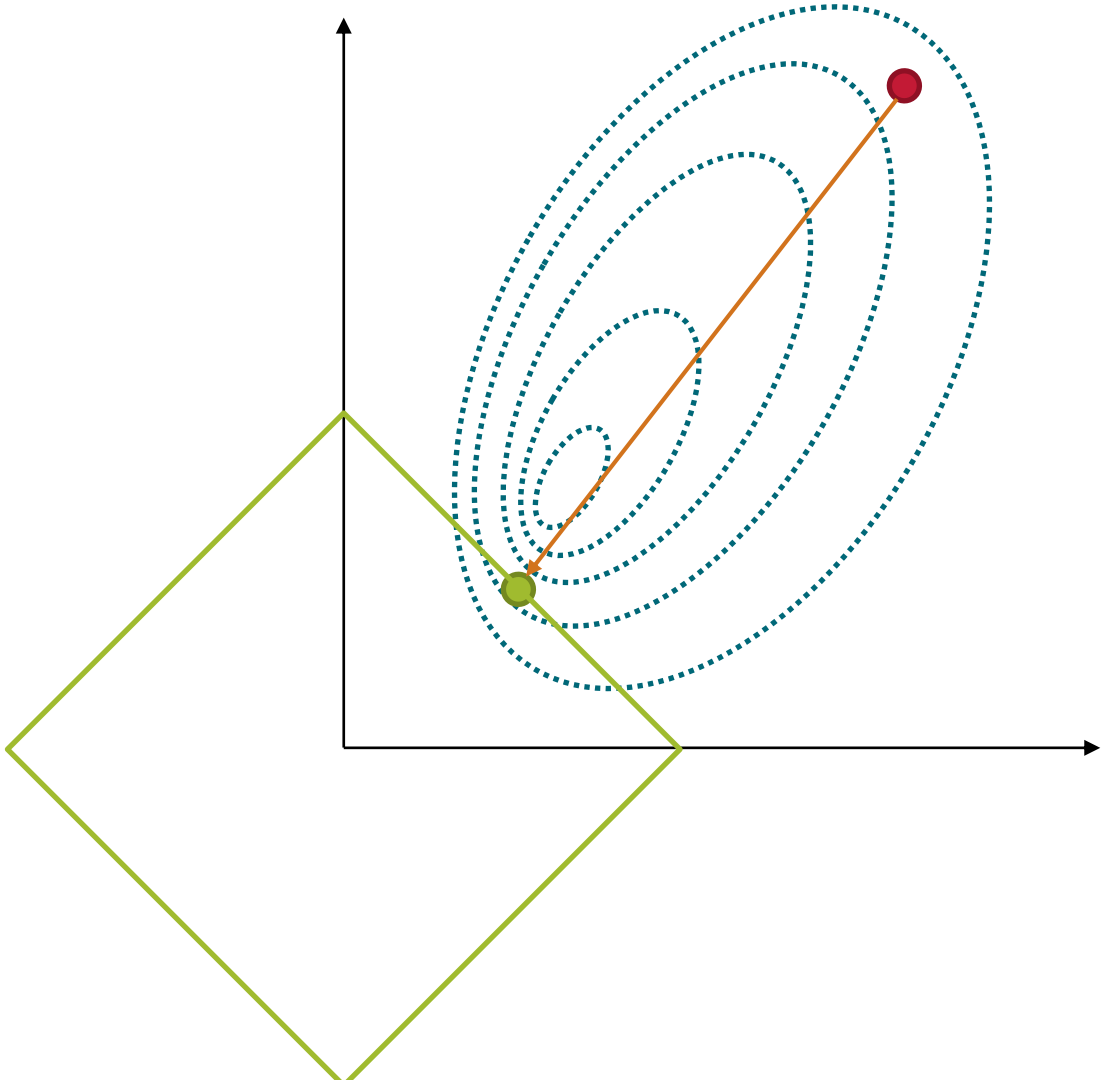
depth, width and paths

global layout

Leverage points: Gradient descent optimization



Leverage points: Regularizer

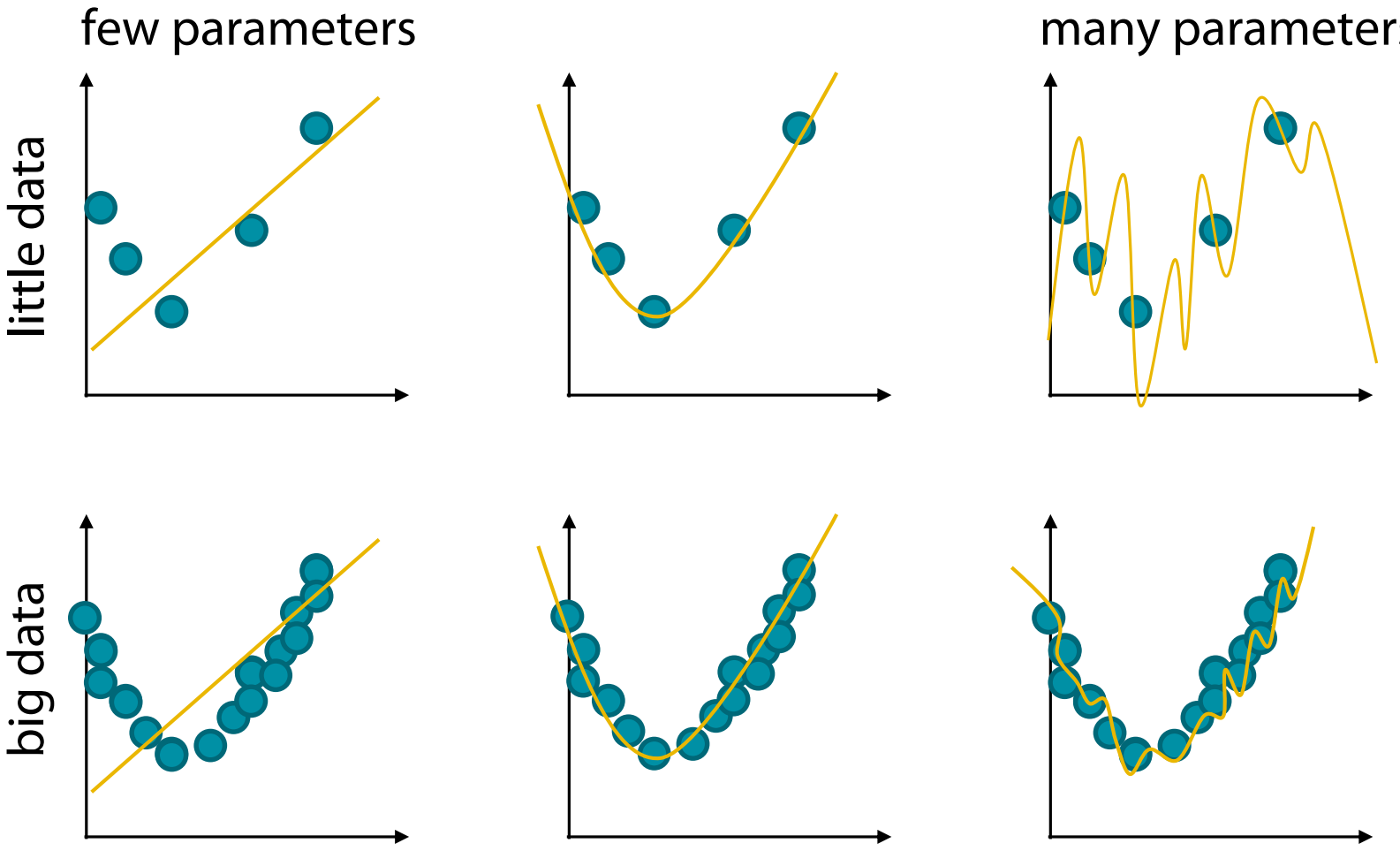


GPU

- Very fast execution of simple calculations in parallel
- Solutions for feed forward and back propagation exist
- By some considered the driving force behind the late advances in deep neural network training
- Example
  - 28x28, 500 mini-batch size, 2 conv layers
  - CPU i7 3.40GHz: *380.28m*
  - GeForce GTX 480: *32.52m*
- Memory is bottleneck (few GB)
- For the future, expect faster GPUs with more memory

Example source: http://deeplearning.net/tutorial/lenet.html

Big Data



Big Data

Problem

- Up to million of parameters
- Repeated passes of the training data
- Tendency to overfit

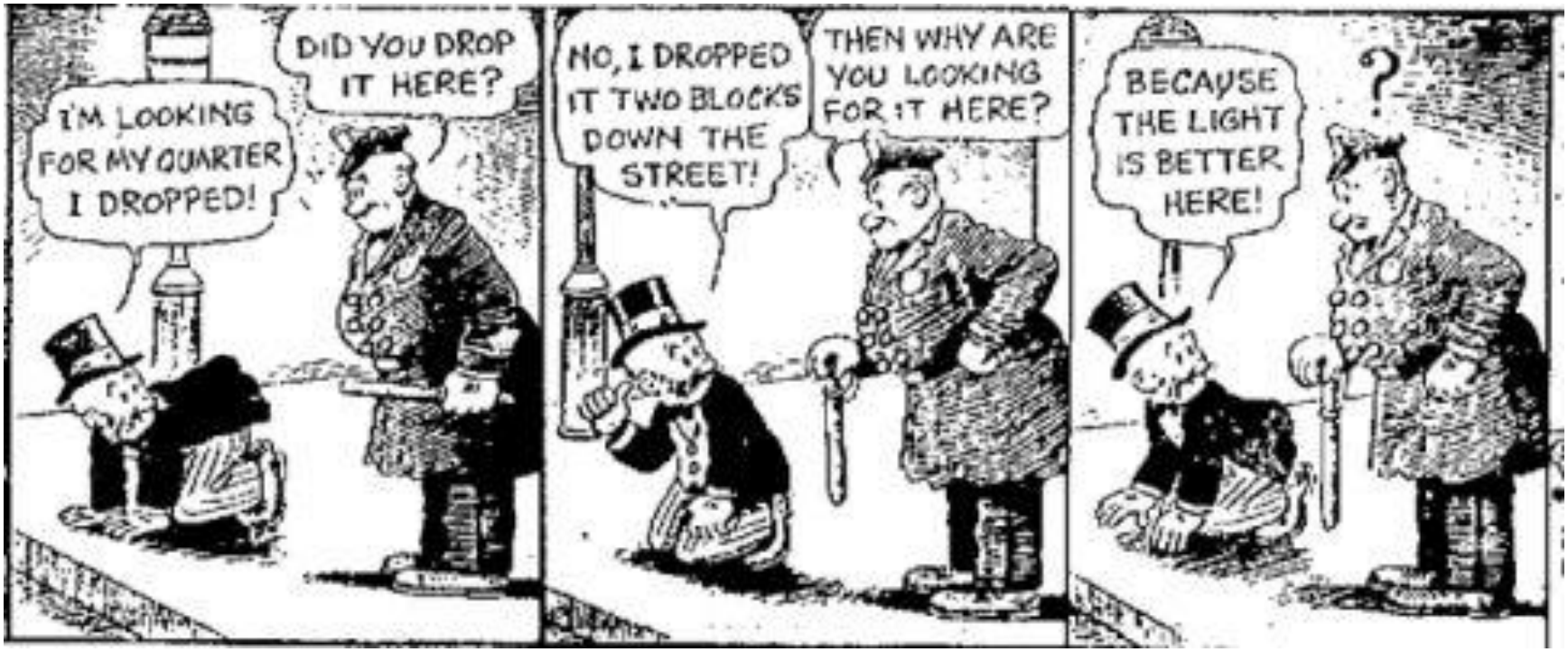
Solution

- Recent developments increase the amount of available data manifold => from 100 to 1.000.000 images
- More data allows for more parameters with less danger of overfitting => larger networks

Dangers of Big Data

- We invent Problems for which we have the data

Big Data



How good is your selfie?

http://karpathy.github.io/2015/10/25/selfie/

CNN

- Suitable for areas where distributed presentations can be found
- More heavily regularized than fully connected (FC) networks
- Less parameters

=> They lessen some of the Deep Learning problems

- Less flexible than FC networks
- Some authors announce the end of CNN in some areas
- FC limited by current processing power

Weight share in CNNs

- Removes the locality constraint from the receptive field
- Allows to detect features regardless of their position in the image
- Acts as a regularizer
- Greatly reduces the complexity

Not always sensible

- E.g. face detection where some feature appear spatial restricted

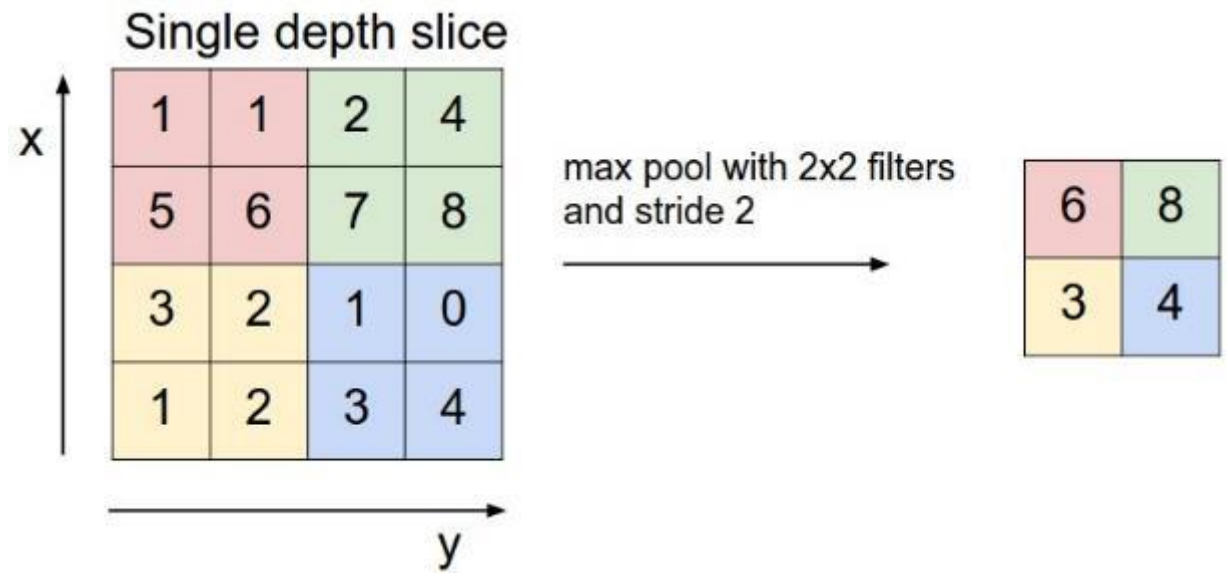
Pooling-Layer

- Special purpose layer to reduce complexity
- Results in faster network convergence
- Enforce generalization through the removal of spatial information (regularizer)

Different types

- Averaging
- L2 norm (square root of summes squares)
- Max-pooling

Max-Pooling



- „The winner takes it all“ => Only the strongest response to a feature passes the layer
- In practice mostly only 2x2 version used
- Lately used less and less (strides have similar effect)[1]

Image source: http://cs231n.github.io/convolutional-networks/

[1] Springenberg et al., “Striving for Simplicity: The All Convolutional Net”, http://arxiv.org/abs/1412.6806

A typical CNN network

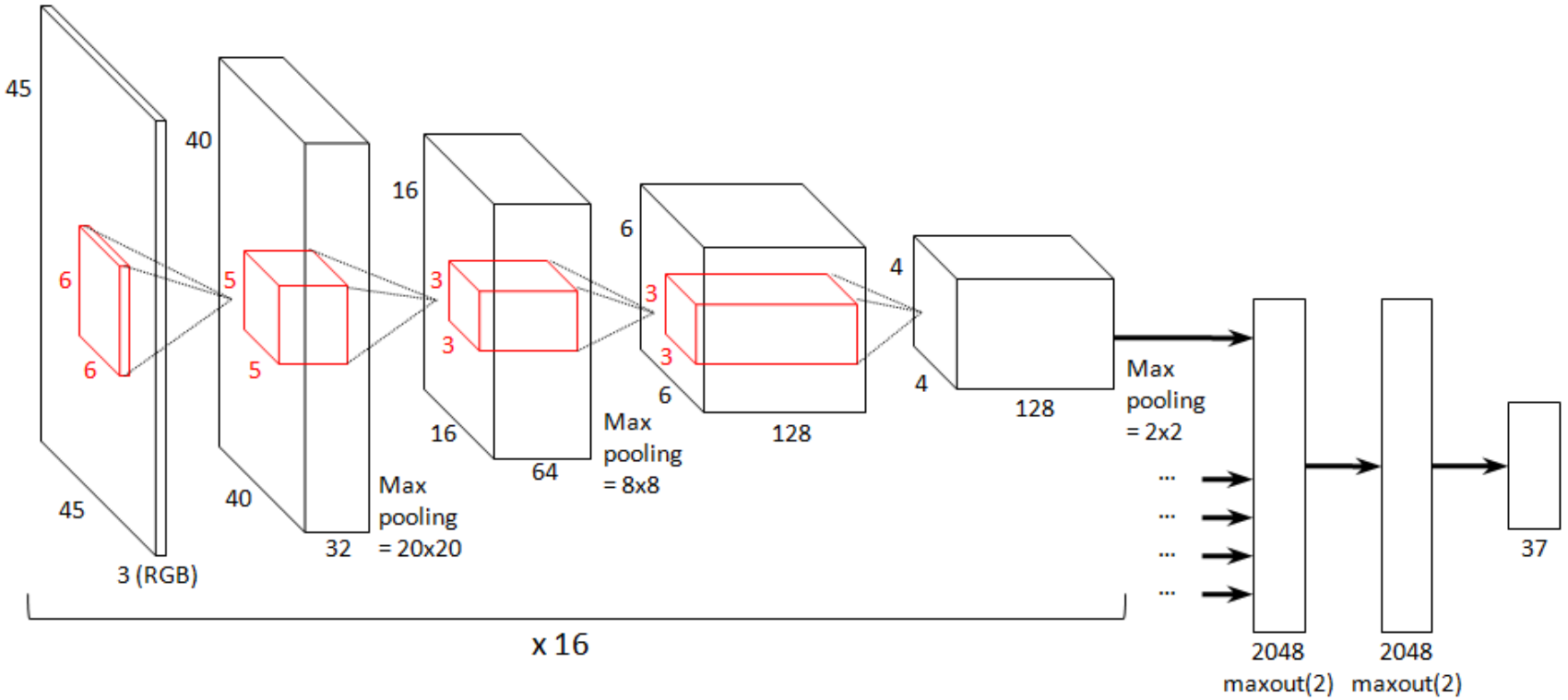
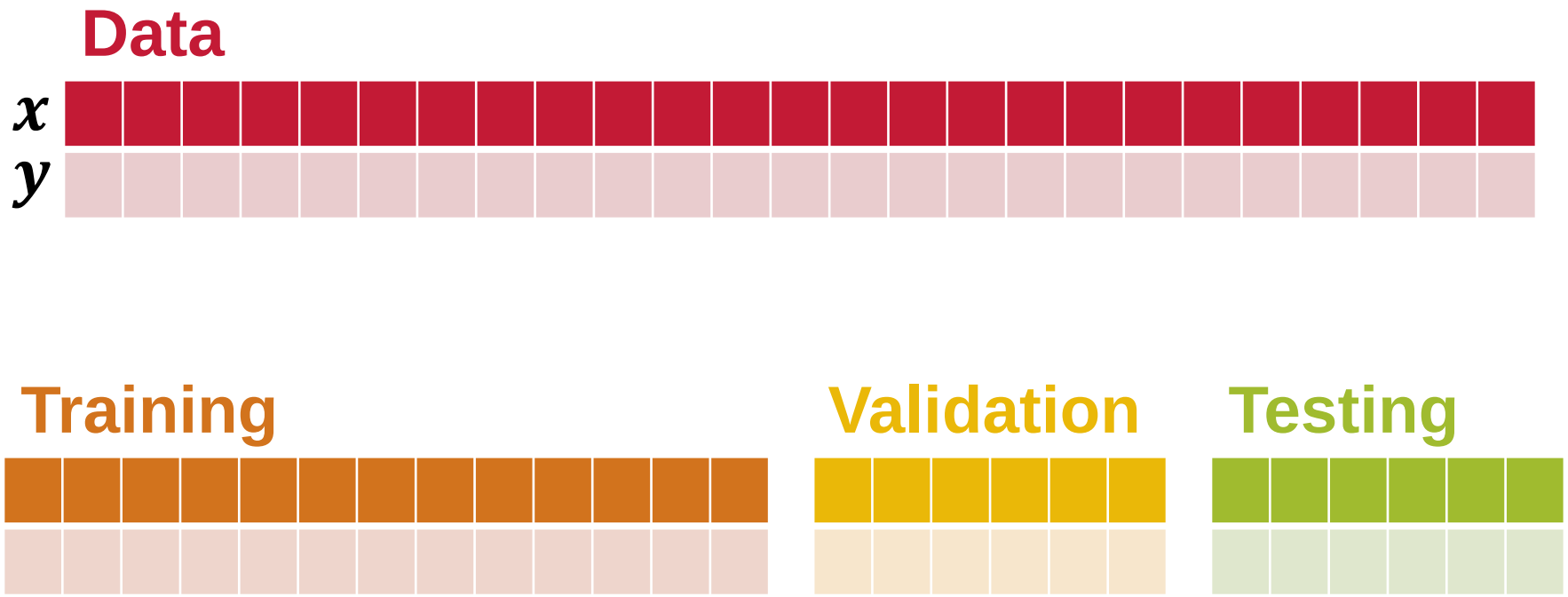


Image source: http://benanne.github.io/images/architecture.png

How to train your network II



Gradient descent tactics

$$w_i' \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

1. Batch
   - gradient of whole training dataset at once
   - huge memory requirements
   - only off-line training
   - guaranteed to converge to global/local minimum
2. Stochastic
   - sample-wise update
   - faster than batch and on-line training
   - frequent updates with high variance
   - overshoots sometimes, therefore usually coupled with steadily decreasing learning rates
3. Mini-batch
   - reduces the variance
   - can use very fast optimized matrix computation
   - batch size often determined by memory (and for GPU power 2)

Principle source on gradient descent subject: http://sebastianruder.com/optimizing-gradient-descent/

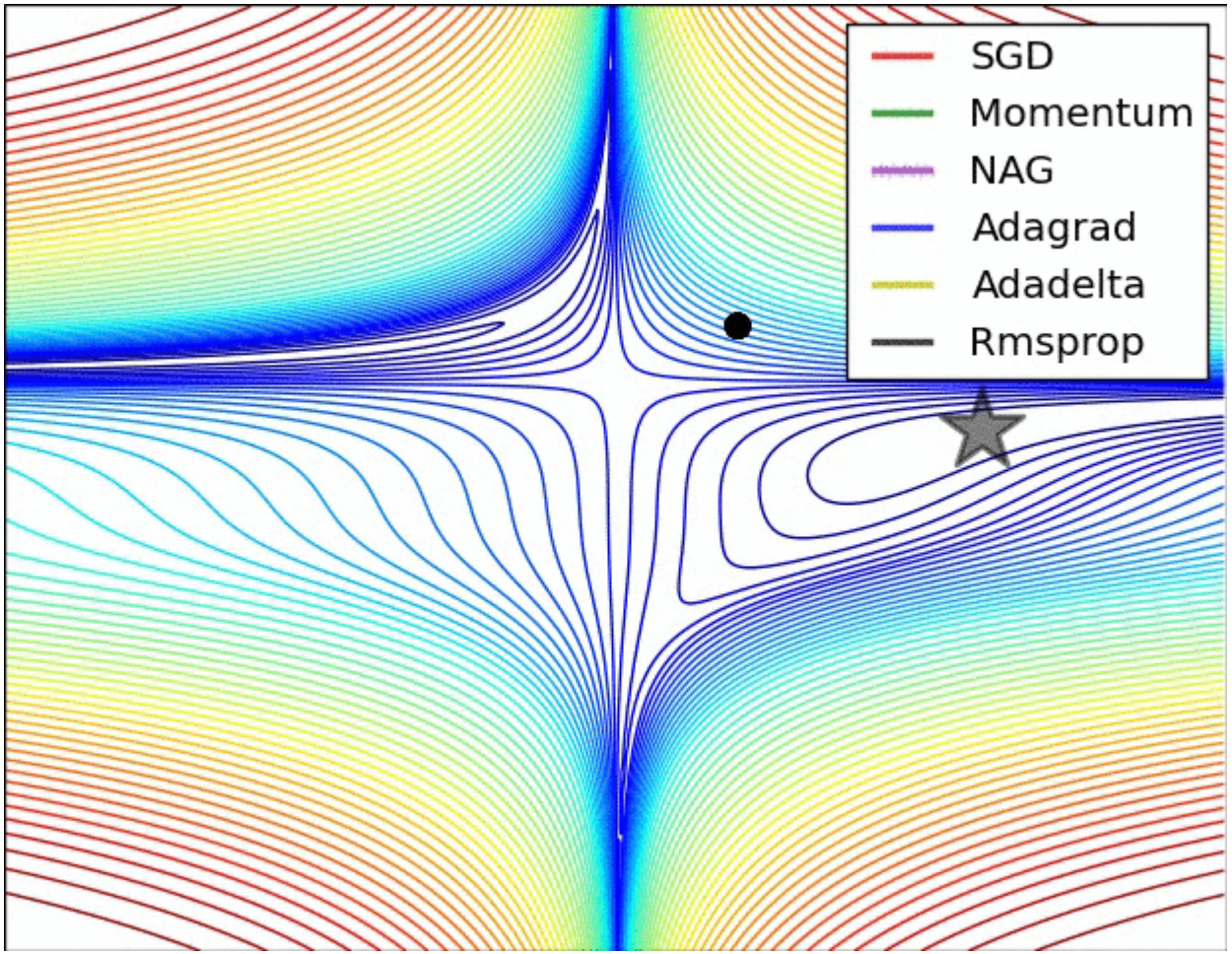
Gradient descent optimization

$$w_i' \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

1. Momentum[1]
   - i.e. speed gain to escape oscillation in ravines
2. Nesterov accelerated gradient[2]
   - Looking ahead slows down approaching the bottom of a valley
3. Adagrad[3]
   - Gives more important to rare features by recording occurrences, monotonically decreasing learning rate, removes the need to set the learning rate
4. Adadelta[4] & RMSprop[5]
   - Like Adagrad, but with short-term memory
5. Adam[6]
   - Like Adadelta/RMSprop but with momentum and bias correction

Articles: See presentation notes

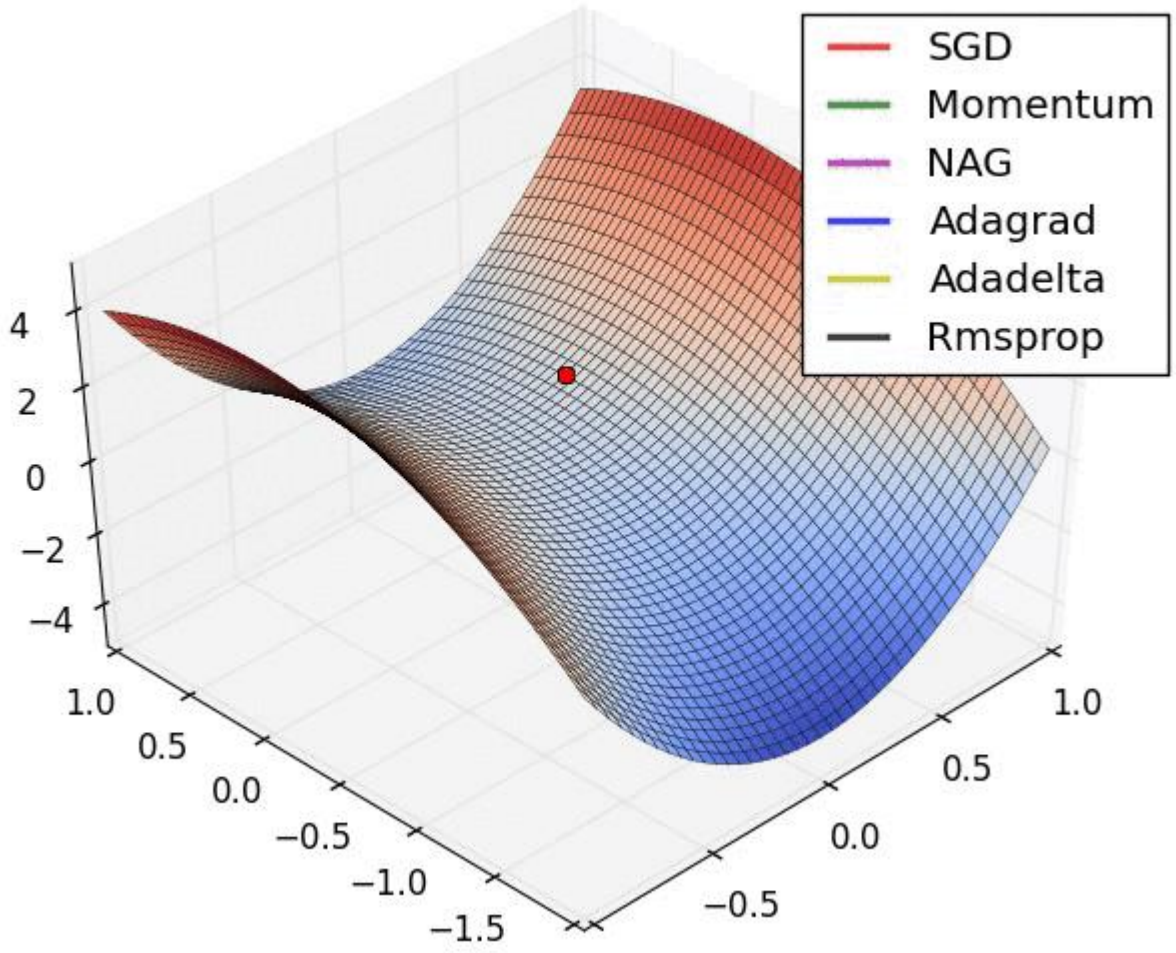
Gradient descent optimization



SGD optimization on loss surface contours

Source: Alec Radford (https://twitter.com/alecrad)

Gradient descent optimization



SGD optimization on saddle point

Source: Alec Radford (https://twitter.com/alecrad)

Gradient descent optimization

- Adadelta, RMSprop or Adam
  - faster
  - converge more reliable
  - have less trouble with complicated structures
  - Do not require the setting of the learning rate
- Adam maybe slightly better due to bias correction and momentum properties[1]
- Some implementation for distributed and parallel SGD exist
- Further information:
  - http://sebastianruder.com/optimizing-gradient-descent
  - http://www.deeplearningbook.org/contents/optimization.html

[1] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

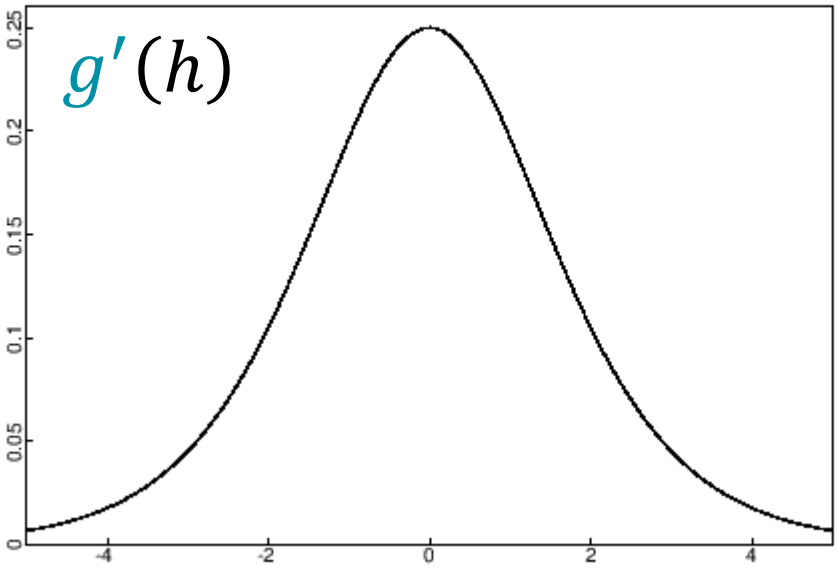
Error and output functions

Sum of square differences

$$E = \frac{1}{2}\sum_k (t_k - y_k)^2$$

$$\frac{\partial E}{\partial w} = (y - t)g'(h)x$$

$$h = \boldsymbol{wx} + b$$



$$g(h) = \frac{1}{1 + e^{-h}}$$

Image source: http://www.billharlan.com/papers/logistic/img39.png
Principle source on error functions: http://neuralnetworksanddeeplearning.com/chap3.html

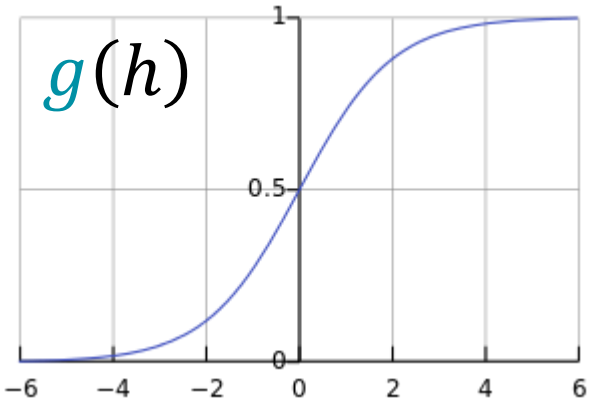
Error and output functions

Cross-entropy

$$E = -\frac{1}{N}\sum_{x}[t \ln y + (1-t)\ln(1-y)]$$

$$\frac{\partial E}{\partial w} = \frac{1}{n}\sum_{x} x_j(g(h) - t)$$

$$h = \boldsymbol{w}\boldsymbol{x} + b$$



$$g(h) = \frac{1}{1+e^{-h}}$$

Image source: https://en.wikipedia.org/wiki/Sigmoid_function
Principle source on error functions: http://neuralnetworksanddeeplearning.com/chap3.html

Error and output functions

Log-likelihood

$$E = -\sum_k^K t_k \ln y_k$$

$$\frac{\partial E}{\partial w_{jk}} = a_j(y_k - t_k)$$

Softmax

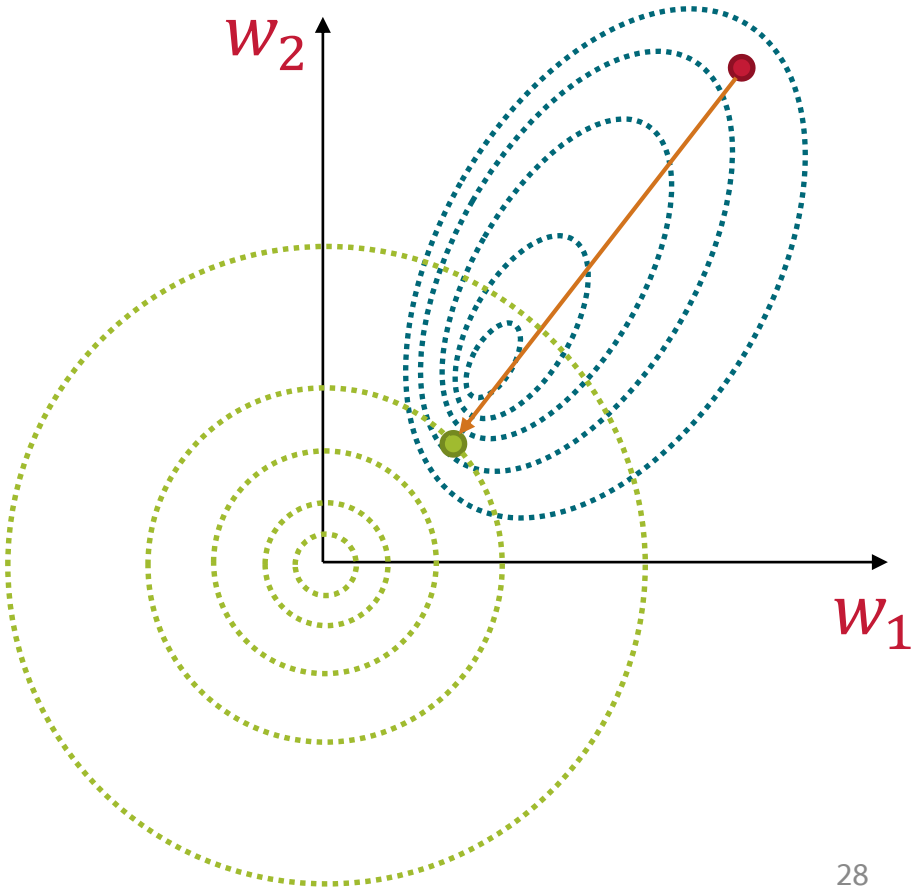
$$g(h_k) = \frac{e^{h_k}}{\sum_l^K e^{h_l}}$$

Principle source on error functions: http://neuralnetworksanddeeplearning.com/chap3.html

L2 regularization or weight decay

$$E = E_o + \frac{\lambda}{2N} \sum_w w^2$$

$$w_i' \leftarrow \left(1 - \frac{\eta\lambda}{N}\right) w_i - \eta \frac{\partial E_0}{\partial w_i}$$

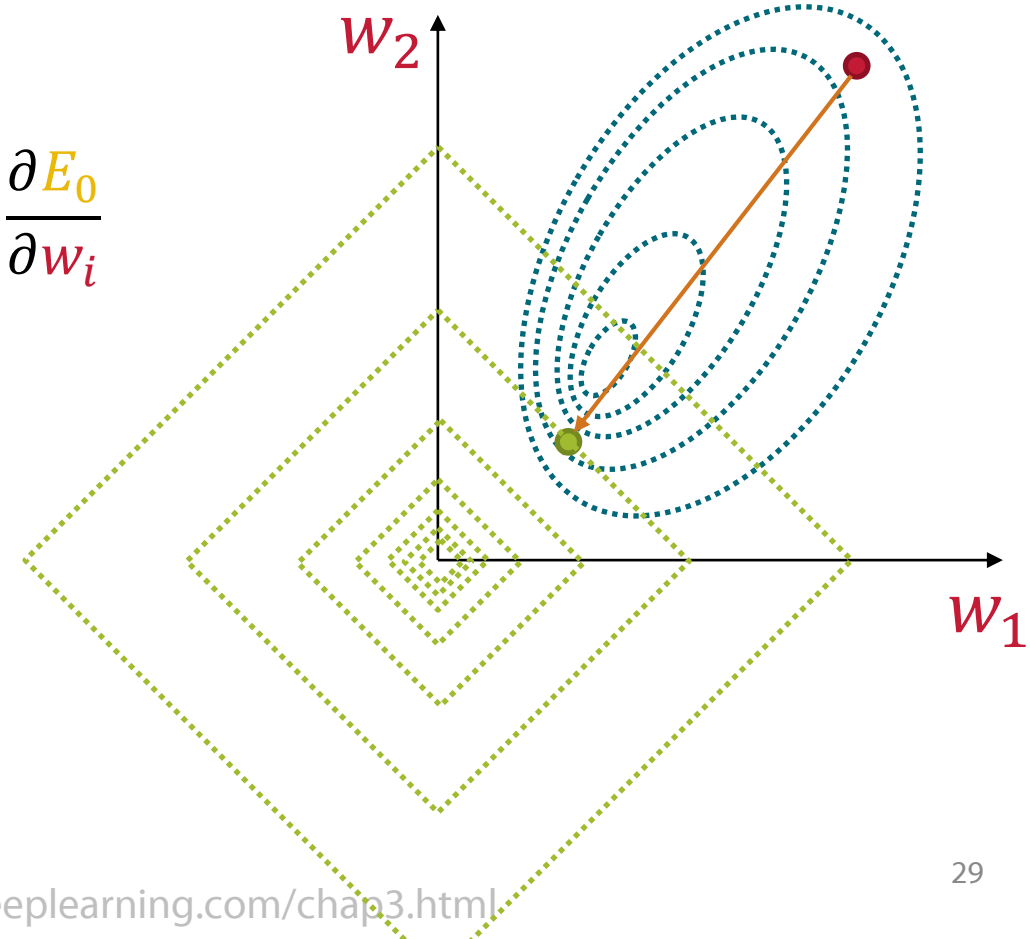


Principle source: http://neuralnetworksanddeeplearning.com/chap3.html

L1 regularization

$$E = E_o + \frac{\lambda}{N} \sum_w |w|$$

$$w_i' \leftarrow w_i - \frac{\eta\lambda}{N} sgn(w_i) - \eta \frac{\partial E_0}{\partial w_i}$$



Principle source: http://neuralnetworksanddeeplearning.com/chap3.html

Regularization: Interlude

- Constraints the model's parameters
- And therefore it's complexity / ability to adapt to the data
- In practice, this leads to simpler models
- And often better generalization

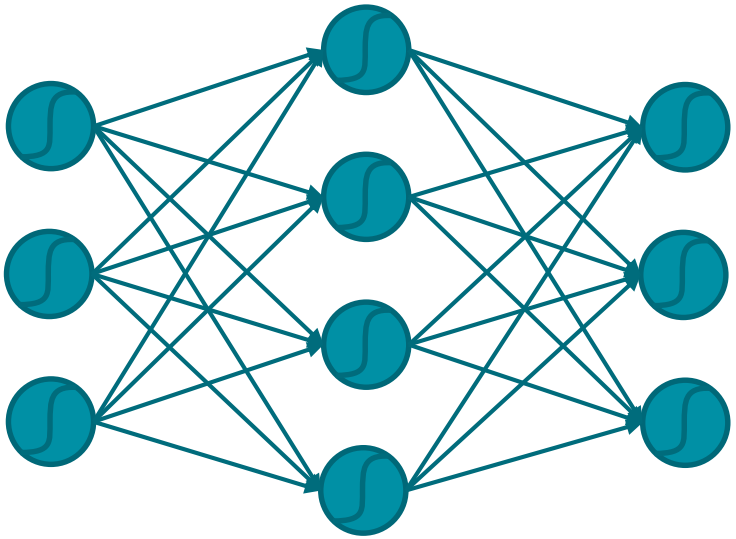
- But they do not have to be right or even better
- They are heuristics tricks and usually poorly understood systematically
- Why does it work? No one really knows.
- Training a DNN is like fitting a 1 Million parameter polynomial to 100.000 samples!

- The greater question behind is: How do we generalize?

Averaging / Ensemble

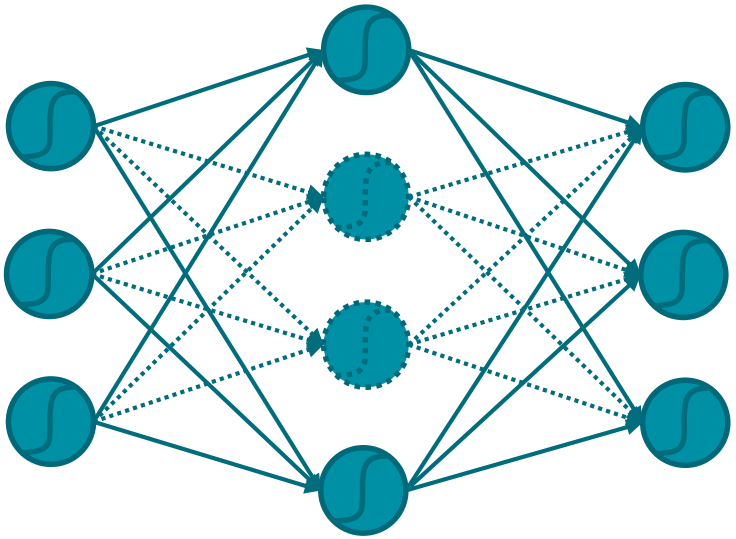
- Assumption: Every network overfits in a different way
- Idea: Letting them vote on the solution (averaging)
- Problem: DNN is expensive

Dropout[1] [hot]

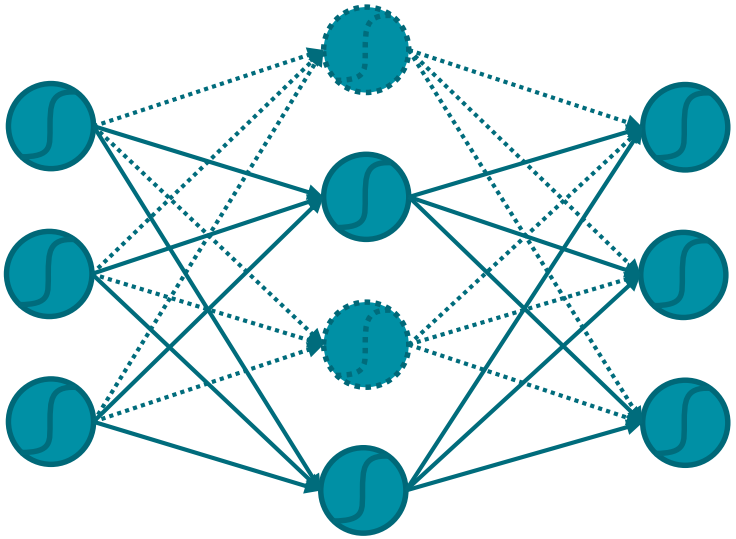


[1] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

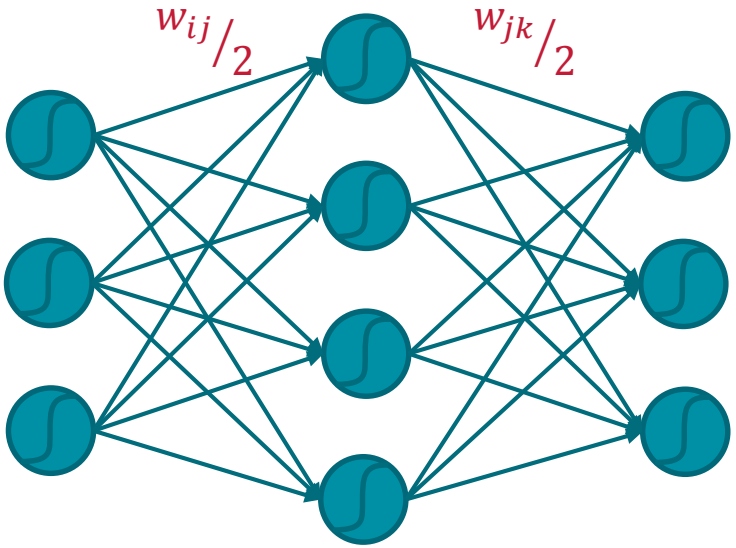
Dropout



Dropout



Dropout



Dropout

- Very successful method against overfitting

- Presumed effect
  - Sub-networks are trained
  - Indirect acts like ensemble methods (many small networks)
  - Approximates the geometric mean of sub-nets (in the case of softmax layer)
  - Second effect is a bagging-like sub-network training
  - Reduces complex co-adaptation of neurons and hence only robust features are used

- Recently considered overcome[1]

[1] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

Data augmentation

- Big Data better than small data
- But often only limit amounts available

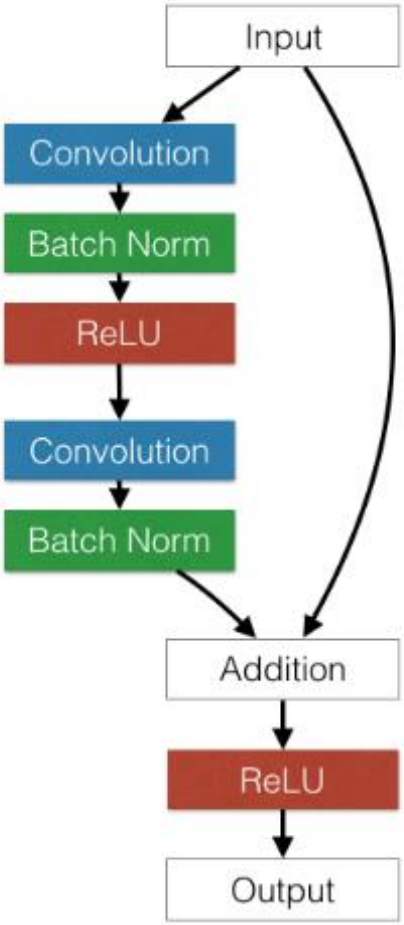
- Idea: Same samples, but with slight plausible variations
- For images these can be:
  - Illumination
  - Slight distortions
  - Rotation & Translation
  - Noise
- More training data, more variations, better generalization

Batch normalization [hot]

- **Observation**: Network trains faster if input is whitened
- **Problem**: Distribution of layer inputs change after each learning step -> requires adaptation, hence low training, saturations
- **Idea**: Batch normalization $a_{i+1} = g(BN(\boldsymbol{W}\boldsymbol{a}_i))$
  - Fix means and variances of layer inputs
  - Once for each mini-batch the means and variance is guessed
  - Then $y^k = \gamma^k \hat{x}^k + \beta^k$, where γ, β are learned and k denotes feature
  - Implemented as special purpose layer (trainable, build into network)
- **Effects**
  - Larger learning rate
  - Less initialization dependence
  - Regularizing (sometimes no Dropout required)
  - ➔ Faster training (up to 14x)

[1] Sergey Ioffe, Christian Szegedy (2015), "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", http://arxiv.org/abs/1502.03167

Batch normalization [hot]



[1] Sergey Ioffe, Christian Szegedy (2015), "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", http://arxiv.org/abs/1502.03167

Other regularizers

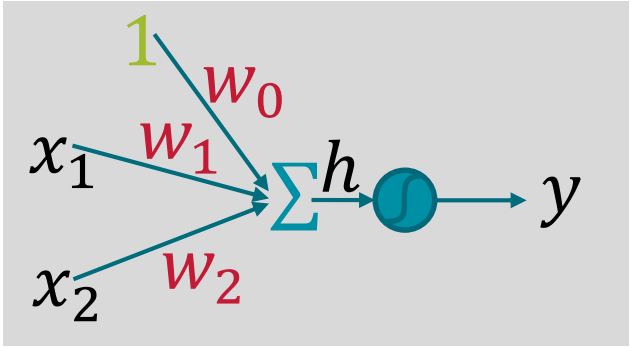
- Add random noise to the network
- Many regularizer terms for the error function
- Multi-task learning

More

http://www.deeplearningbook.org/contents/regularization.html

Weight initialization

Classical approach: $W \sim N(0,1)$



Initialization determines:

- Start point of gradient descent
- Local minima we reach
- How fast the network converges

Problem: Neuron saturation might occur in the hidden layers

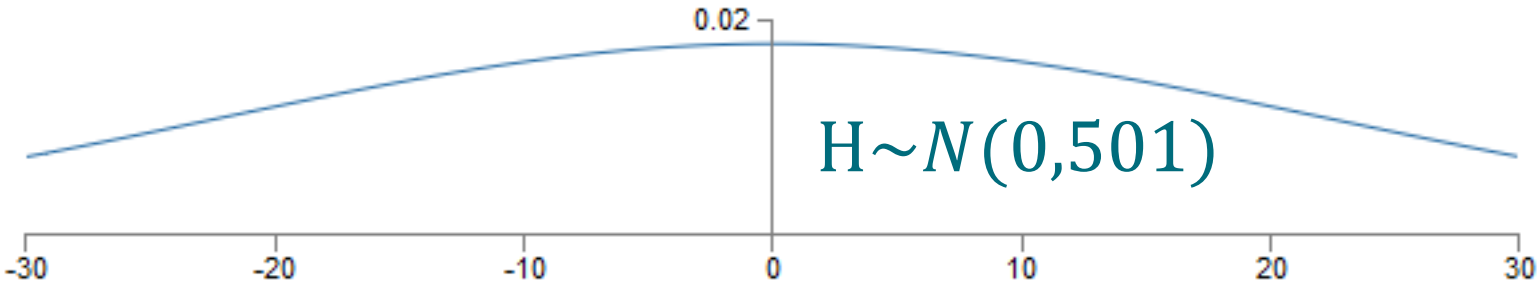


Image source: http://neuralnetworksanddeeplearning.com/chap3.html

Weight initialization

Weighted Gaussian solution: $W_j \sim N(0, 1/N_j)$

- Where N_j is the number of input to neuron j

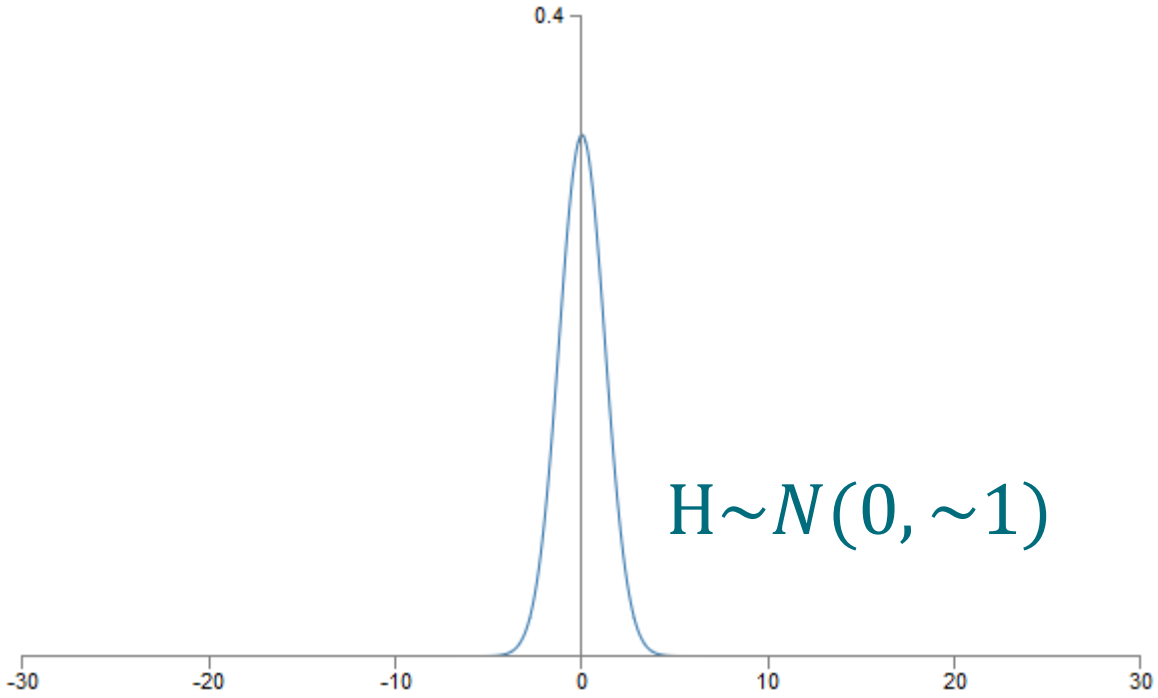
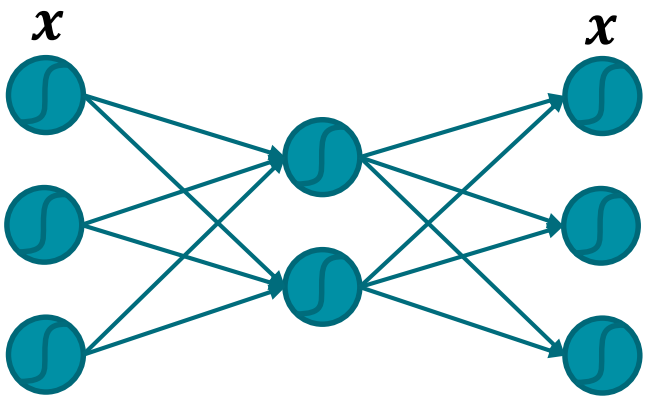


Image source: http://neuralnetworksanddeeplearning.com/chap3.html

Pre-training[1]

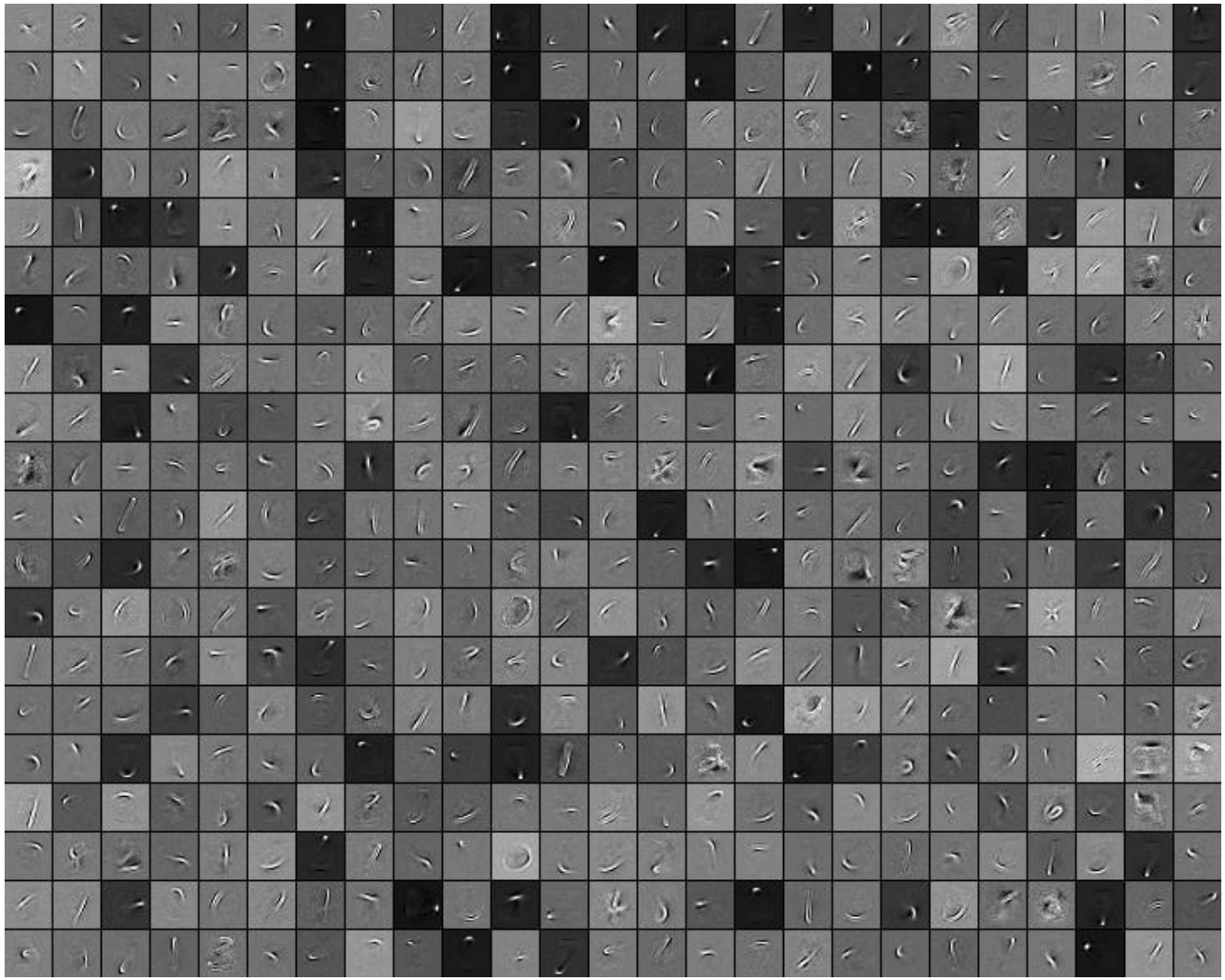
- Unsupervised training of reduced representations (greedy i.e. layer-wise)
- Then few iterations of supervised
- Concentration on „what's important“
- Acts similar to de-noising



- Only works with deep believe networks (RBMs) and stacked auto-encoders
- Slow and only FC

[1] Hinto et al. (2006) “. "A fast learning algorithm for deep belief nets.“ Neural computation 18.7: 1527-1554

MNIST pre-training example

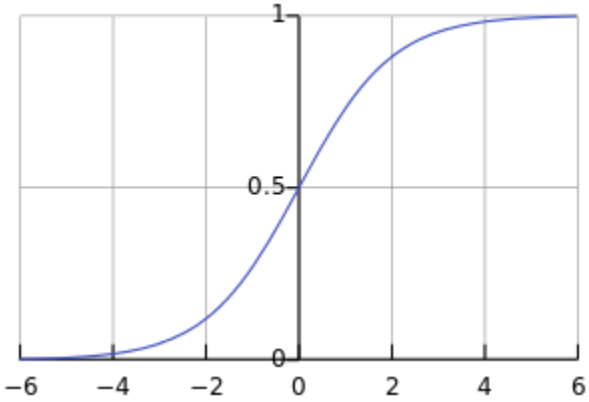


Source: http://www.acsu.buffalo.edu/~devansha/blog_why_deep_learning_works.html

Activation functions

Sigmoid

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



Tanh

$$g(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + \hat{e}^{-x}}$$

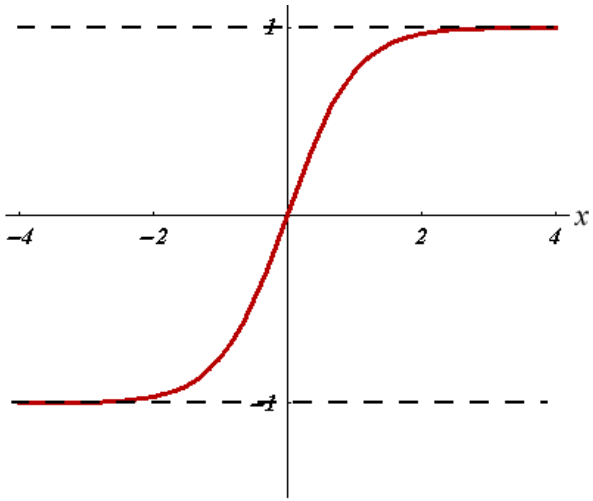


Image sources:
https://en.wikipedia.org/wiki/Sigmoid_function
http://www.efunda.com/math/hyperbolic/display.cfm?name=tanh

Activation functions [hot]

Rectified Linear Units (ReLUs)[1][2]

$$g(x) = \max(0, x)$$

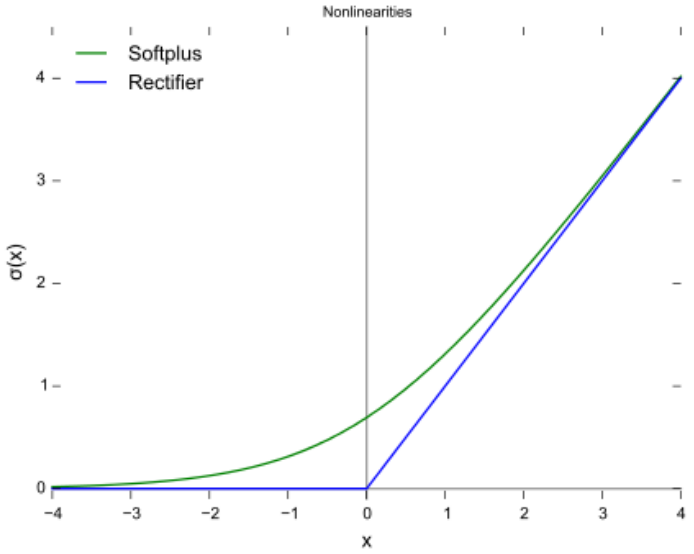


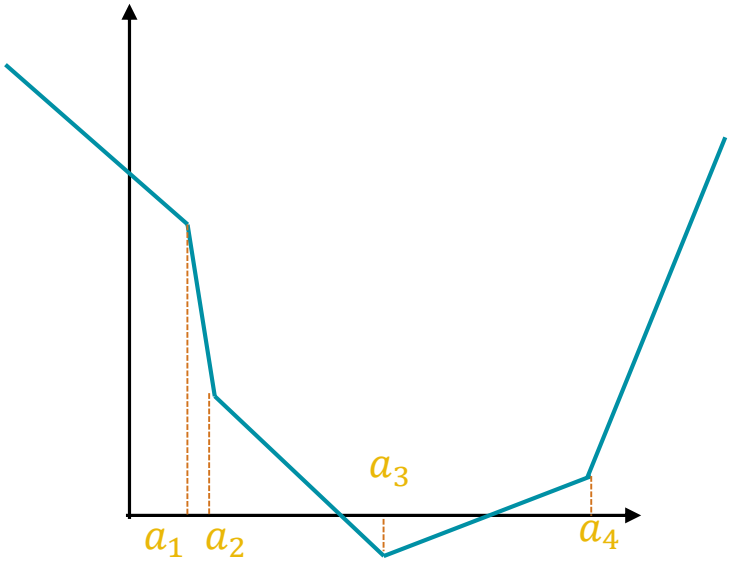
Image source: https://en.wikipedia.org/wiki/Rectifier_%28neural_networks%29

[1] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010
[2] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." International Conference on Artificial Intelligence and Statistics. 2011

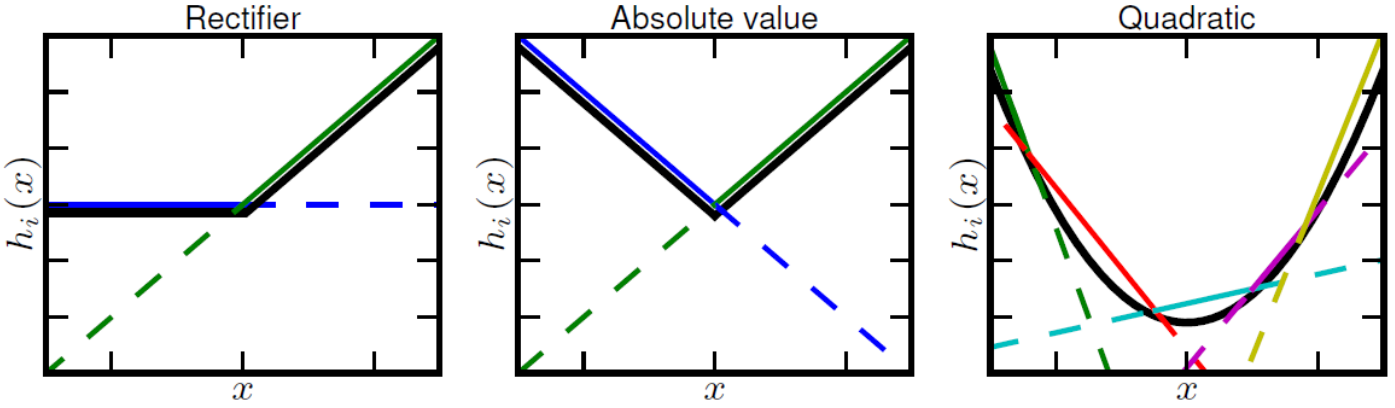
Activation functions [hot]

Maxout[1]

$$g(x) = \max_K z_k$$

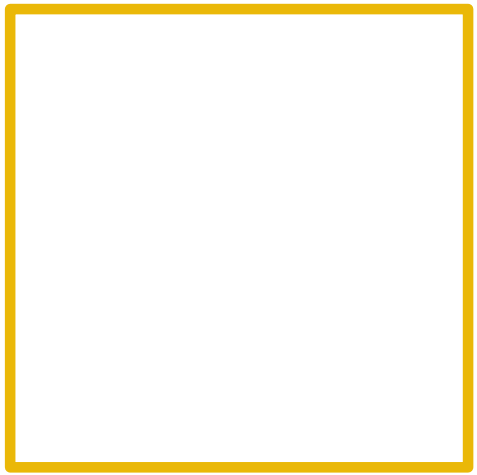


With $z_k = x^T W_k + b_k$ and K is the number of linear models

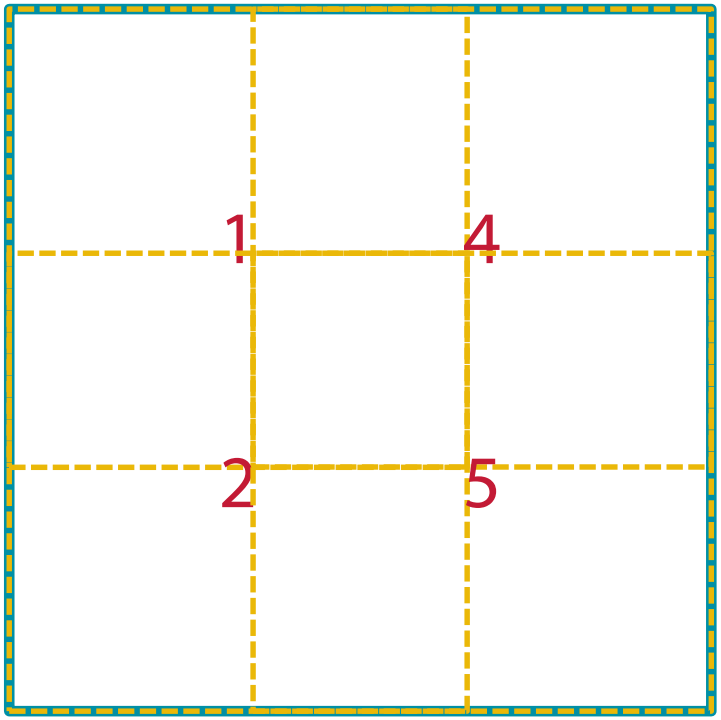


[1] Goodfellow et al., (2013) „Maxout Networks“, https://arxiv.org/abs/1302.4389

FC <-> CNN [hot]



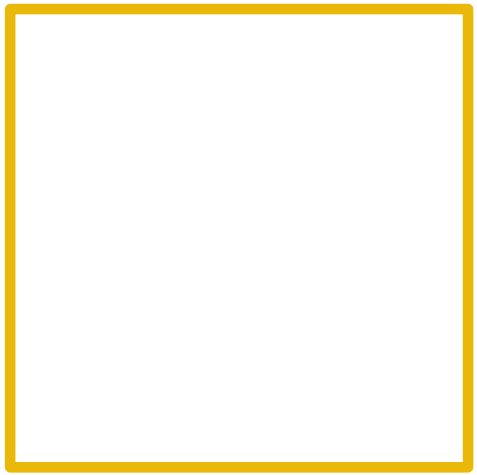
224x224
reduced to 7x7xD
i.e. by 32



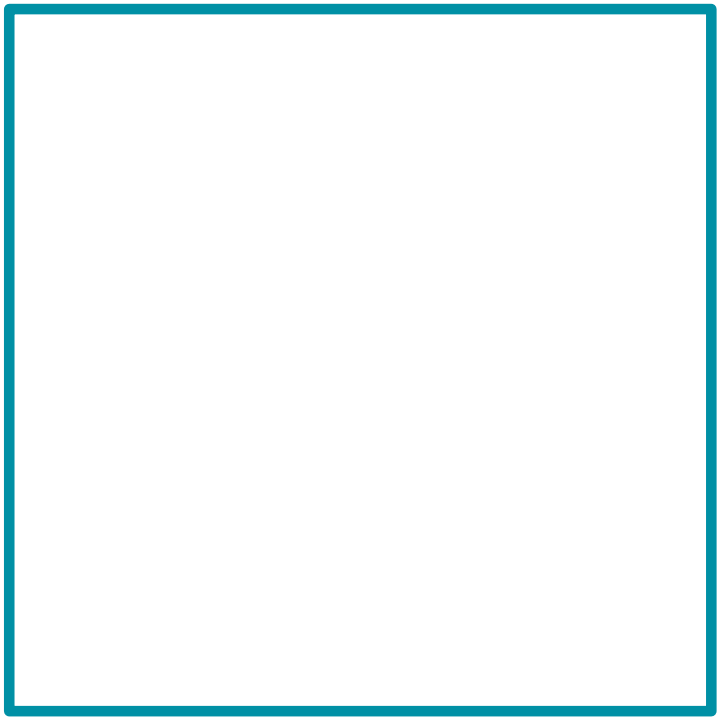
256x256
four applications with
stride 32
four answers

Inspiration: http://cs231n.github.io/convolutional-networks/#convert

FC <-> CNN



224x224
reduced to 7x7xD
i.e. by 32



256x256
one application
four dimensional answer

[1,4,2,5]

Dilated convolution[1]

Idea

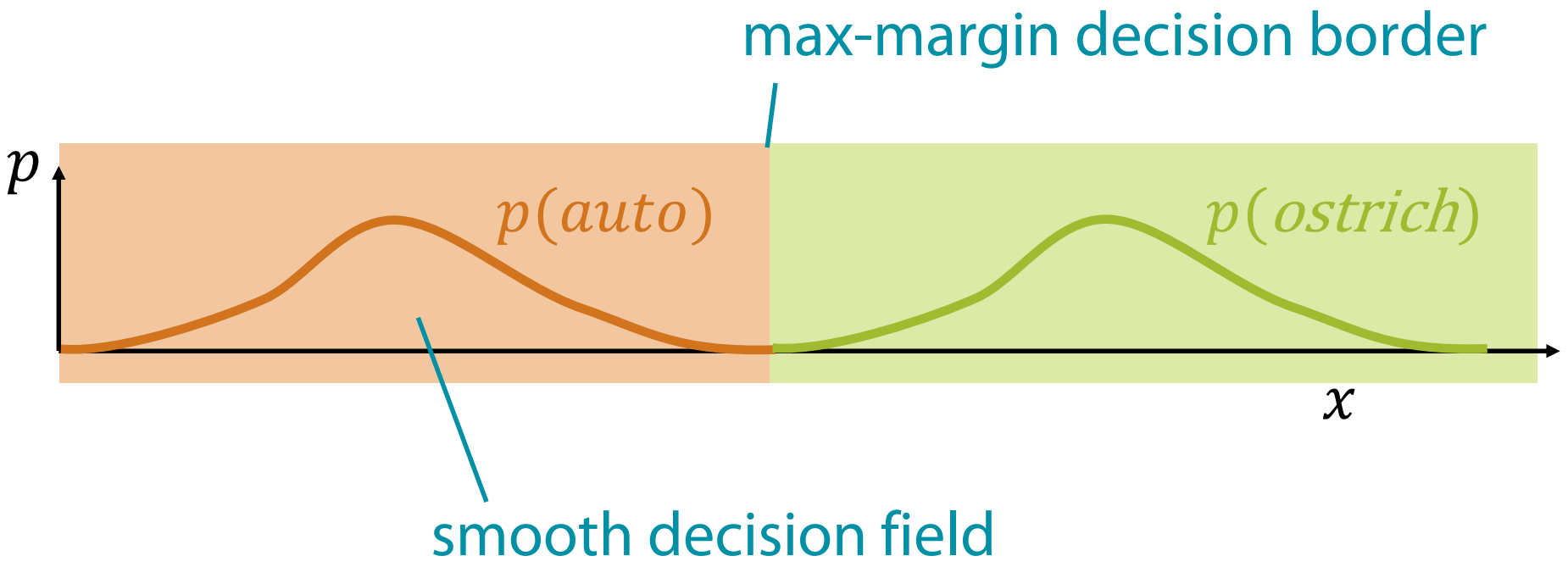
- Use sparse kernels
- Always together with dense kernels
- Increases the receptive field of the first layers

Effect

- aggressive merging of spatial information
- faster convergence

[1] Fisher Yu, Vladlen Koltun, (2016) "Multi-Scale Context Aggregation by Dilated Convolutions", https://arxiv.org/abs/1511.07122

The limits of DNN



The limits of DNN

Source: Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013), "Intriguing properties of neural networks"

The limits of DNN

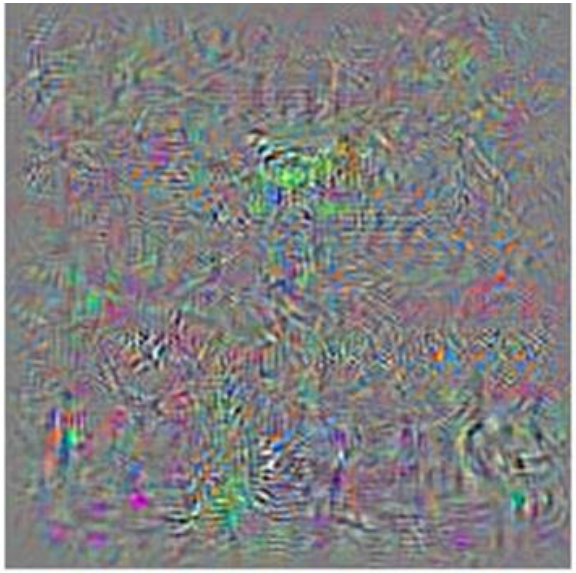
auto

ostrich

Source: Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013), "Intriguing properties of neural networks"

The limits of DNN

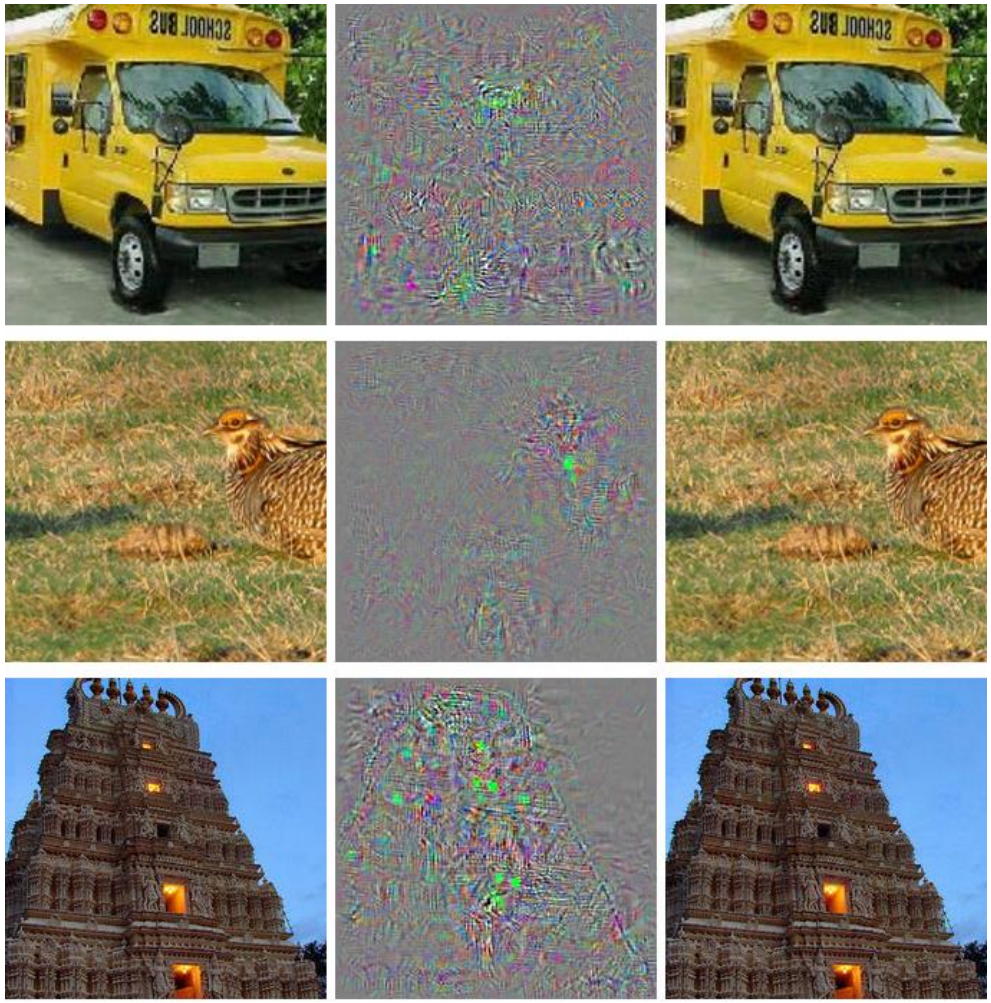
auto



ostrich

Source: Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013), "Intriguing properties of neural networks"

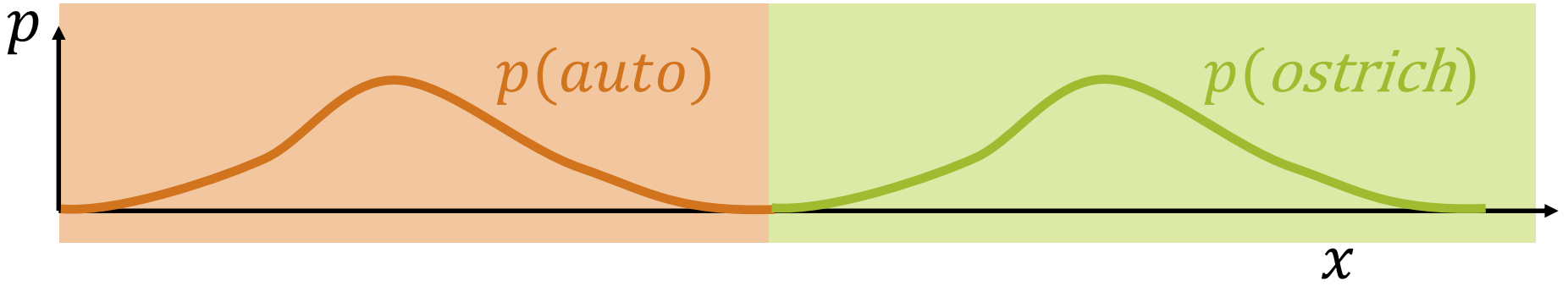
The limits of DNN



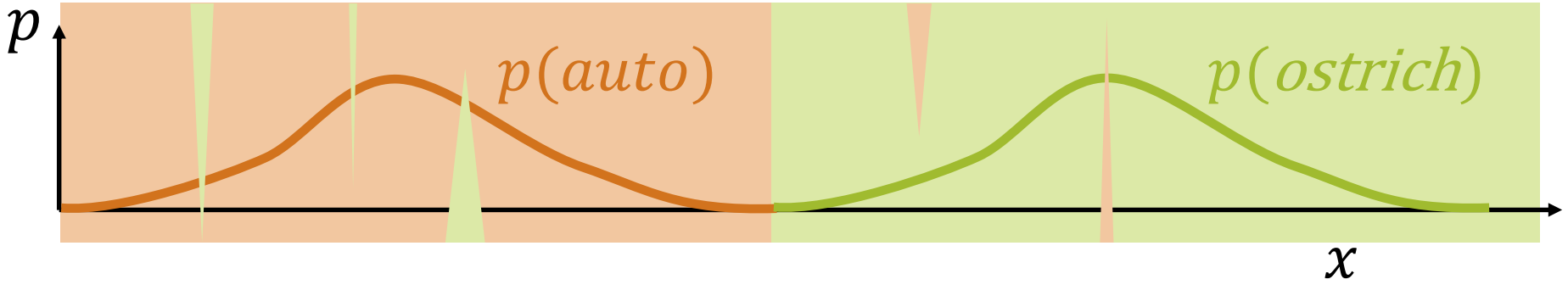
Source: Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013), "Intriguing properties of neural networks"

The limits of DNN

Old assumption



New assumption

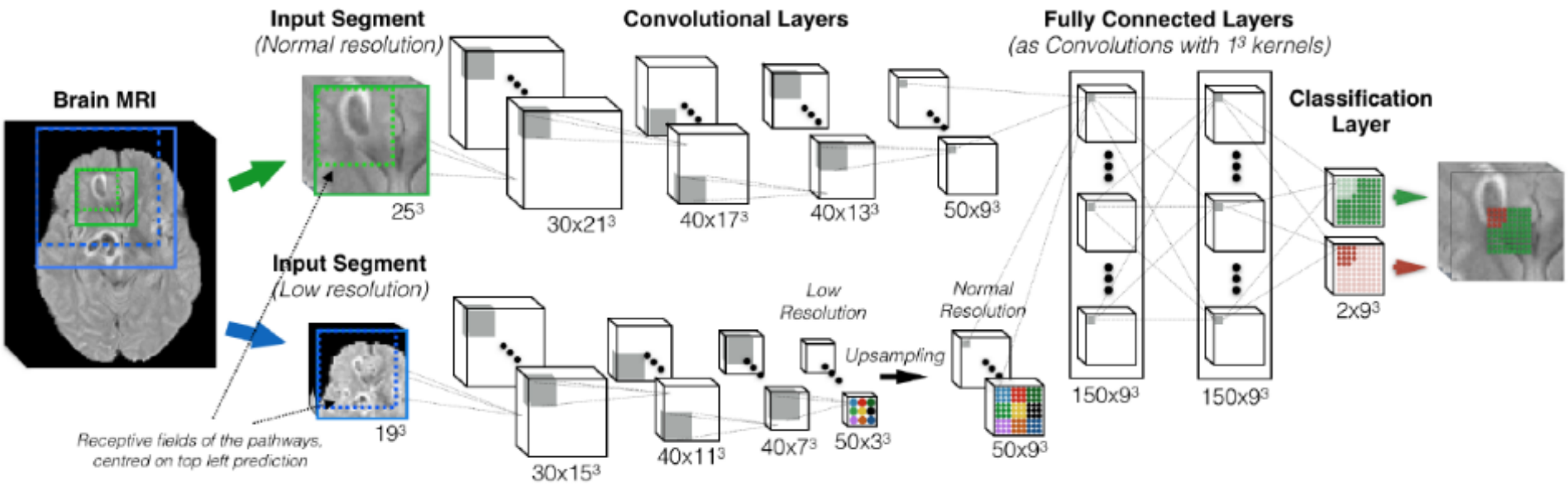


Adversarial samples

- Constructed by selecting the input sample x' nearest to x, for which a neurons activation pattern changes significantly
- Adversary images pose also difficulties for DNNs trained on different training sets, albeit less
  - ➢ There is a pattern behind it
- Adversarial samples dense but narrow (will seldom occur in nature)
- To avoid them nevertheless, adversarial training has been proposed to remove the pockets explicitly[1]

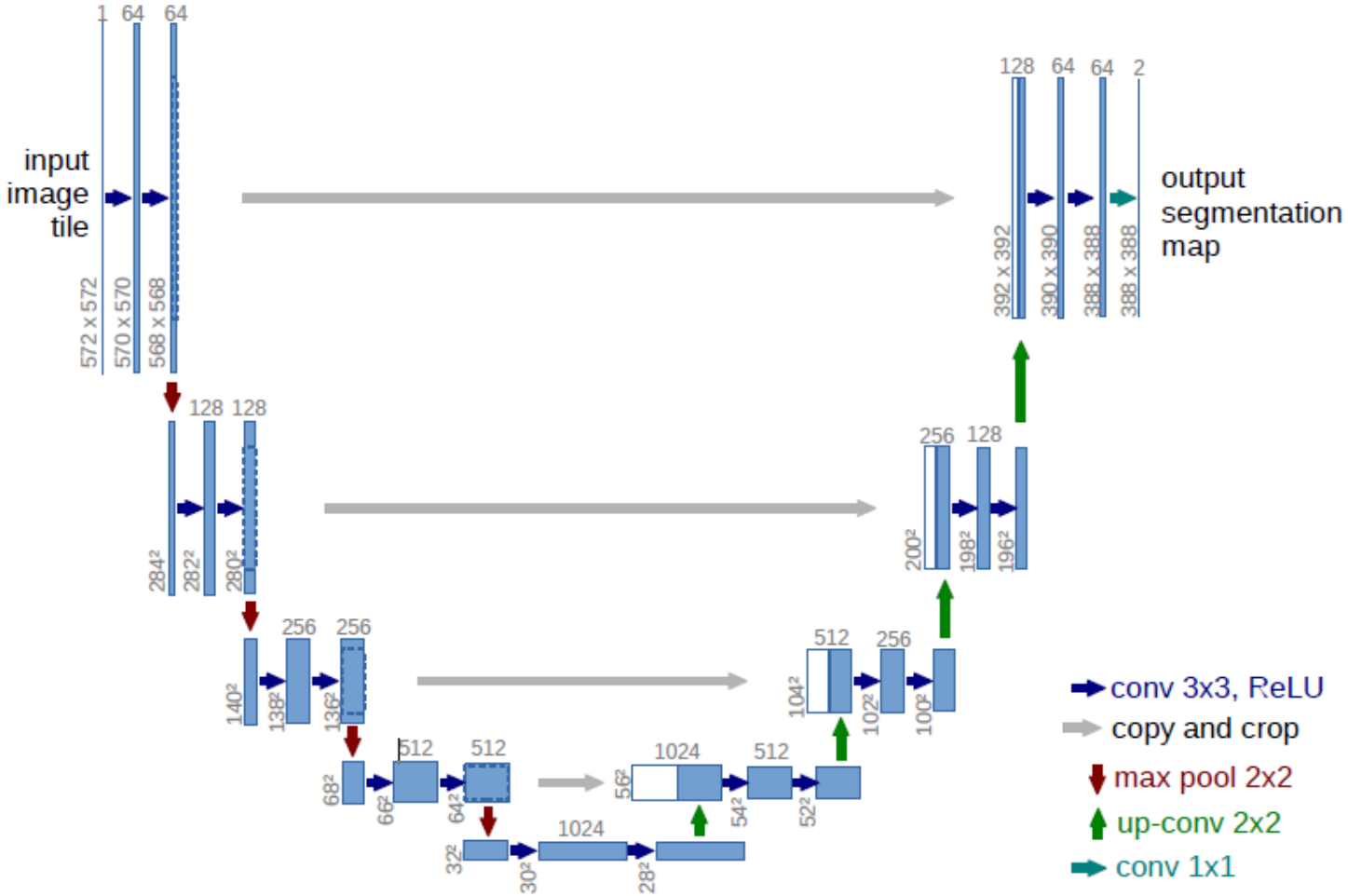
[1]: Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2015), "Explaining and harnessing adversarial examples." ICLR 2015

Multi-resolution networks [hot]



[1]: Kamnitsas et al. (2016), "Efficient Multi-Scale 3D CNN with Fully Connected CRF for Accurate Brain Lesion Segmentation", http://arxiv.org/abs/1603.05959

U-Nets [hot]



[1] Olaf Ronneberger, Philipp Fischer, Thomas Brox, (2015) „U-Net: Convolutional Networks for Biomedical Image Segmentation", http://arxiv.org/abs/1505.04597

Residual Nets [hot]

- From layer l to $l + a$, we want to learn mapping $x_{l+a+1} = H(x_l)$
- Instead, we learn $\mathrm{F}(\mathrm{x_l}) = H(x_l) - x_l$
- Assumption is that the residual function $\mathrm{F}(\mathrm{x_l})$ is easier to learn

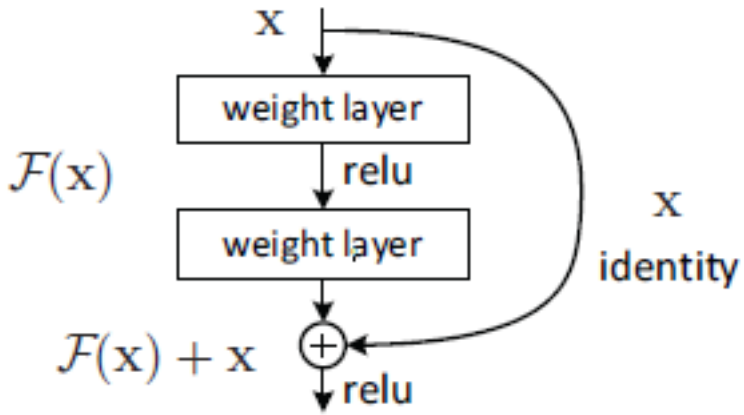
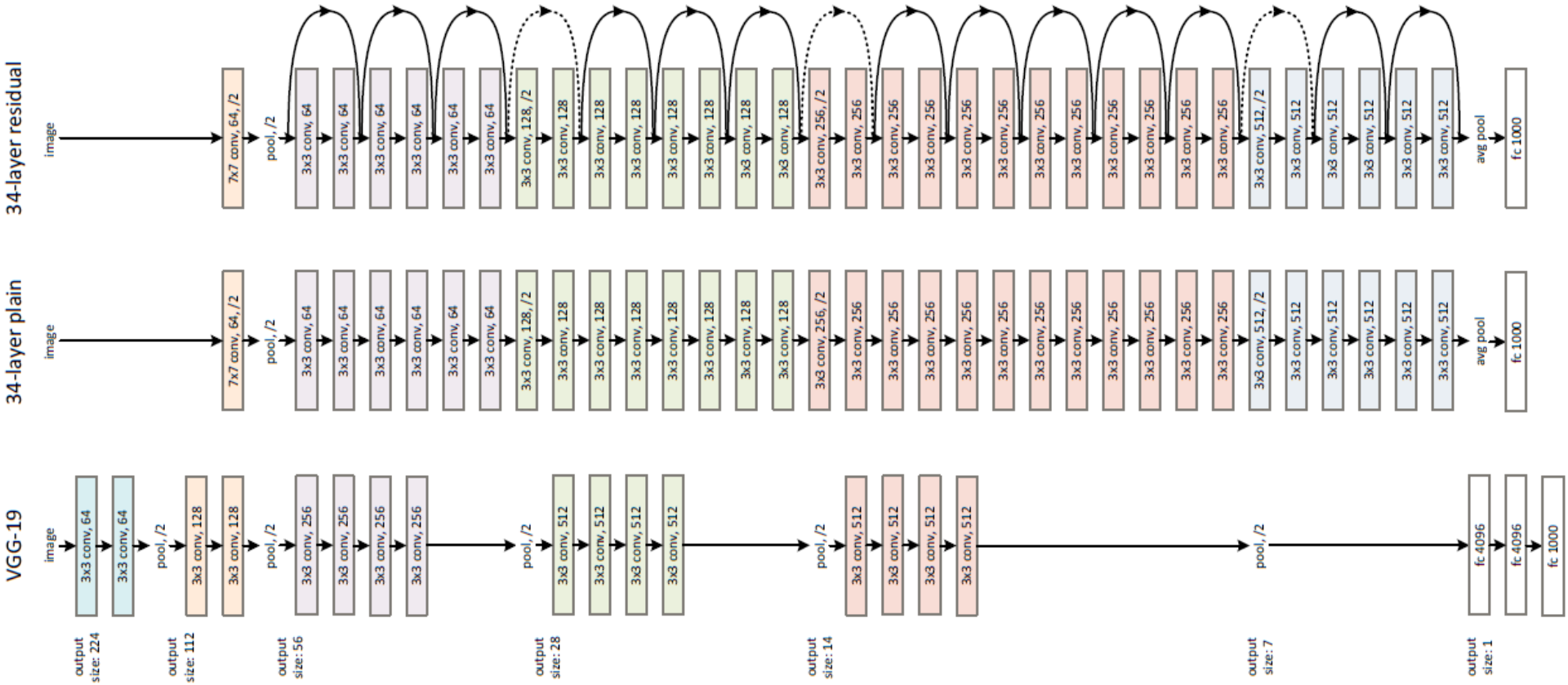


Figure 2. Residual learning: a building block.

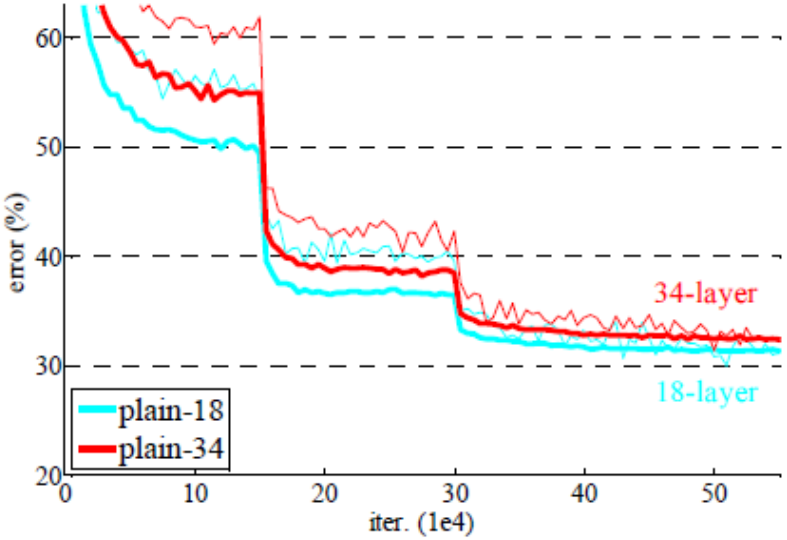
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, (Dec. 2015) „Deep Residual Learning for Image Recognition", http://arxiv.org/abs/1512.03385

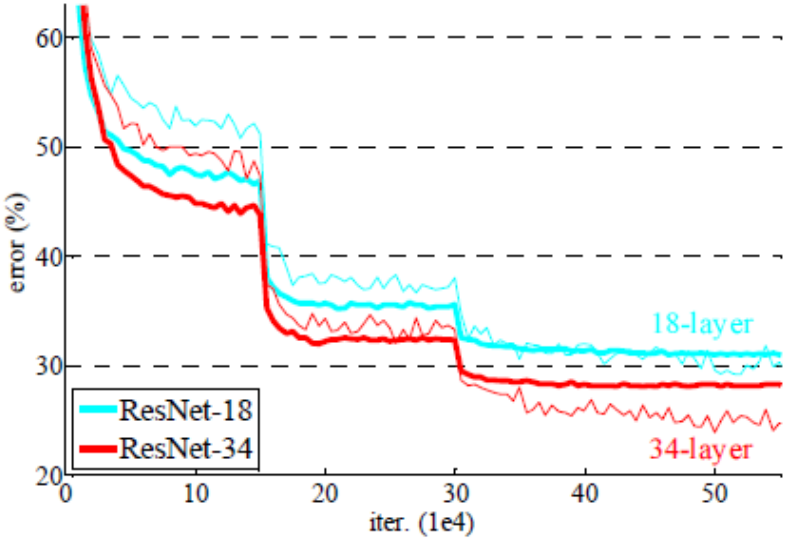
Residual Nets



[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, (Dec. 2015) „Deep Residual Learning for Image Recognition", http://arxiv.org/abs/1512.03385

Residual Nets





model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

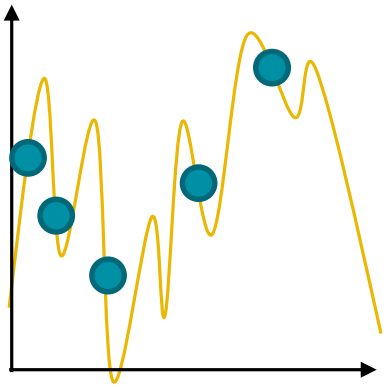
[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (Dec. 2015), „Deep Residual Learning for Image Recognition", http://arxiv.org/abs/1512.03385

Other networks

- Recurrent NNs
  - Notion of dynamic change over time
  - Speech and natural language processing
  - Video processing
  - Near to our universal understanding of algorithms: Can be Turing machines interpreting code!
  - (Google Android voice recognition uses RNNs)
  - Most techniques from FF-ANNs can be employed for RNNs, too
    - BP, Gradient Descent, Regularizers, etc.
- Long short-term memory units
  - Gradient instability affects RNN even worse (the longer they train)
  - Learns conditional gate deciding whether a values should pass, be remembered and/or be forgotten
- Deep belief networks
  - Generative model
  - Can do un- and semi-supervised learning
  - Compositions of restricted Boltzman machines or auto-encoders

Problems with deep networks

1. Amount of parameters
2. Overfitting
3. Gradient vanishing/explosion
4. Input vanishing/explosion



$w < 1$
$w > 1$

Conclusion: Many problems

- Some of them application dependent

Why can we train deep networks?

1. ConvLayers reduce complexity
2. Regularization techniques avoid overfitting
3. ReLUs or Max-Out allow for faster training (up to 3-5)
4. Using GPUs and being patient
5. Careful architectural choices
6. Suitable weight initialization
7. Expanding training set
8. Larger training set
9. Identity connections

➔ Simple set of ideas, powerful if used in concert

Is all that necessary?

- Pure FC network: 2.5k→2k→1.5k→1k→0.5k→10
- 12.1 million parameters

- Only data augmentation, otherwise no gimmicks
- Simply processing power and slowly decreasing learning rate
- 0.35% Error on MNIST-10k (Current best 0.21%)

→ Big Data and great processing power

[1]: Dan Claudiu Cireșan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber (2010), "Deep, Big, Simple Neural Nets Excel on Handwritten Digit Recognition", http://arxiv.org/abs/1003.0358

Is all that worth it?

Accuracy	Method
98.04	FC shallow small + regularizer
99.37	FC shallow wide (100 neurons) + regularizer
97.80	Simple CNN (1 ConvLayer+maxpool, 3 maps, sigmoid, no regularizer)
98.78	More feature maps (now 20)
99.06	Second ConvLayer+maxpool
99.23	ReLUs
99.37	Augmenting training data (shifts only)
99.46	Wider FC Layer (Extra FC layer leads to 99.43)
99.60	Dropout (in FC)
99.67	Ensemble (5 networks)
99.65	Pure FC from before

Deep learning: The solution to all problems?

1. Incredibly successful!
2. Potential!
3. Overcame the standstill in ANN development!

- Segmentation, Translation, Prediction, Dreaming…
- And all of this with self-learning, out-of-the-box tools!
- Too good to be true? Possibly.

Tuning is the key and the curse

Discussion: Current state, future and impact

Keywords

- Where‘s the money?
- Half-life period of new methods
- Theoretical bedrock

- For which applications does it actually excel?
- How much better is DNN than other approaches?
- How well is deep learning understood?

- Medical image processing idiosyncrasies

Background readings

Lectures by Nando de Freitag @ Oxford

- http://www.cs.ox.ac.uk/teaching/courses/2014-2015/ml/
- https://www.youtube.com/playlist?list=PLE6Wd9FR--EfW8dtjAuPoTuPcqmOV53Fu

Lectures of CS231 @ Stanford

- http://cs231n.github.io
- https://www.youtube.com/playlist?list=PLrZmhn8sSgye6ijhLzlIXiU9GNaIwbF8B

Michael Nielsen: Neural Networks and Deep Learning

- http://neuralnetworksanddeeplearning.com (Very good!)

Goodfellow, Bengio and Courville: Deep Learning

- http://www.deeplearningbook.org (Very scientific!)

Andrew Ng: UFLDL Tutorial @ Stanford

- http://ufldl.stanford.edu/tutorial/

Further readings

- Matthew D Zeiler, Rob Fergus (2013), „Visualizing and Understanding Convolutional Networks", https://arxiv.org/abs/1311.2901
- A webpage on visualizing the lower layer activation by creating artificial images that would lead to a maximum neuron activation: http://yosinski.com/deepvis

DNN libraries

- Torch: http://torch.ch
- Caffe: http://caffe.berkeleyvision.org
- Theano: http://deeplearning.net/software/theano
- Tensor Flow: https://www.tensorflow.org
- Keras: http://keras.io

- List of results over the years on popular CV datasets: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html
- Benchmark of networks and architectures: https://github.com/soumith/convnet-benchmarks

Online playthings

- Draw a number and see the CNN recognize it: http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- Observe a network classifying image live in your browser: http://cs231n.stanford.edu/
- Train a DNN live in your browser with JavaScript: http://cs.stanford.edu/people/karpathy/convnetjs/

Fun facts

DNN plays Atari games

http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html

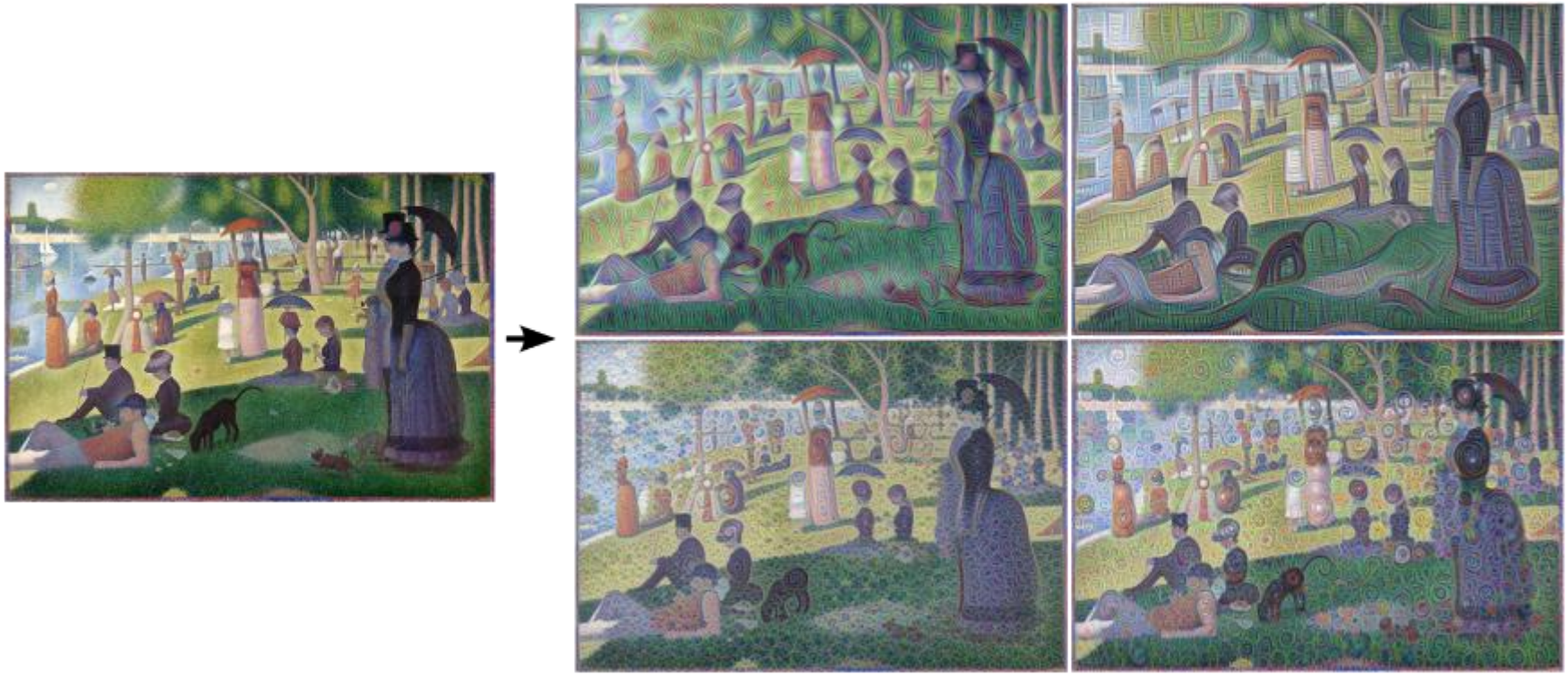
Input: monitor pixels

Output: actions

Learning: re-enforced

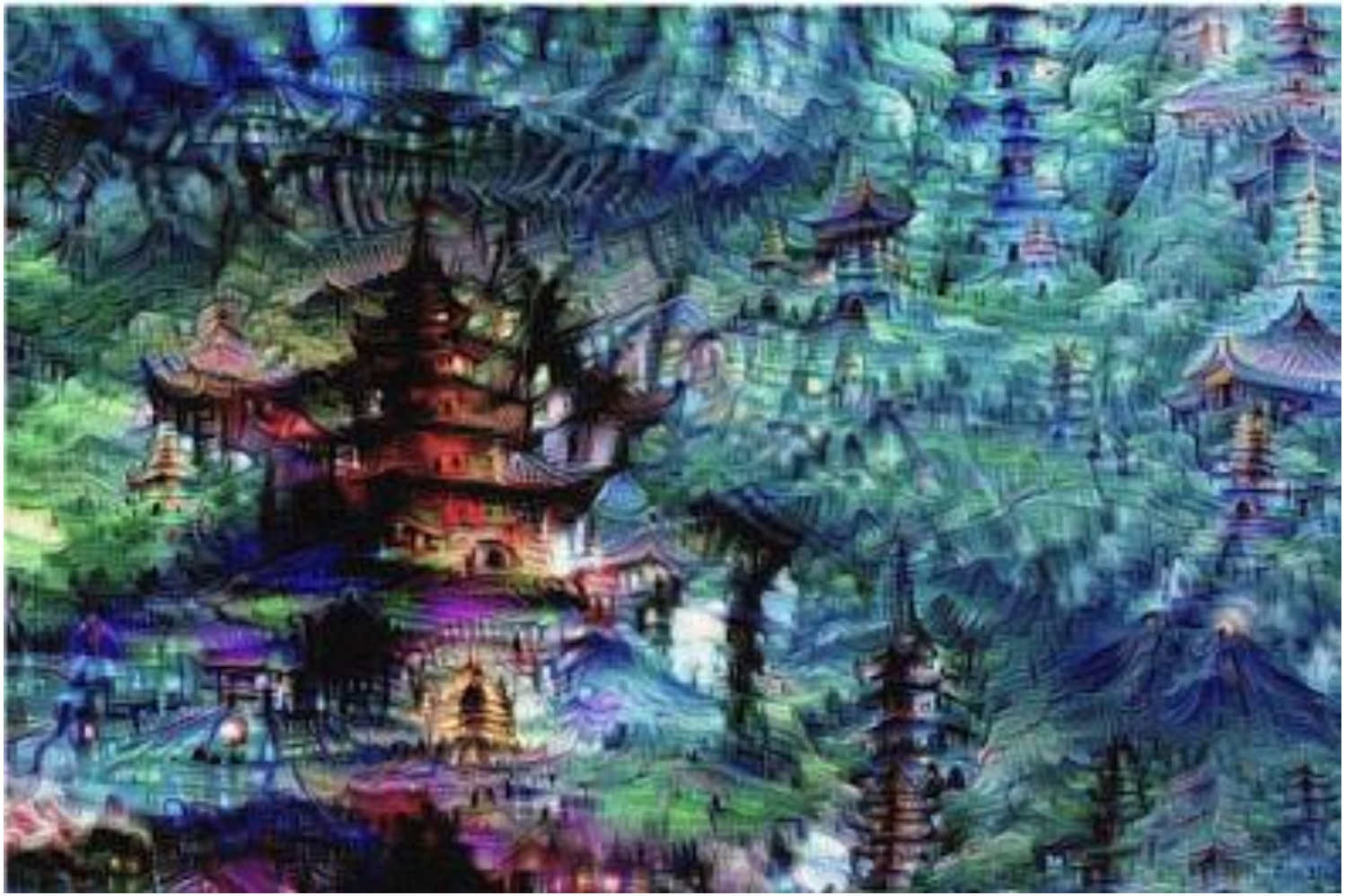
- Learned to play 5 older games
- In three of them „better“ than humans
- No knowledge about the game rules used, all just infered

Inception



Source: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

DeepDream



Source: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Hyperparameter Tips for CNNs

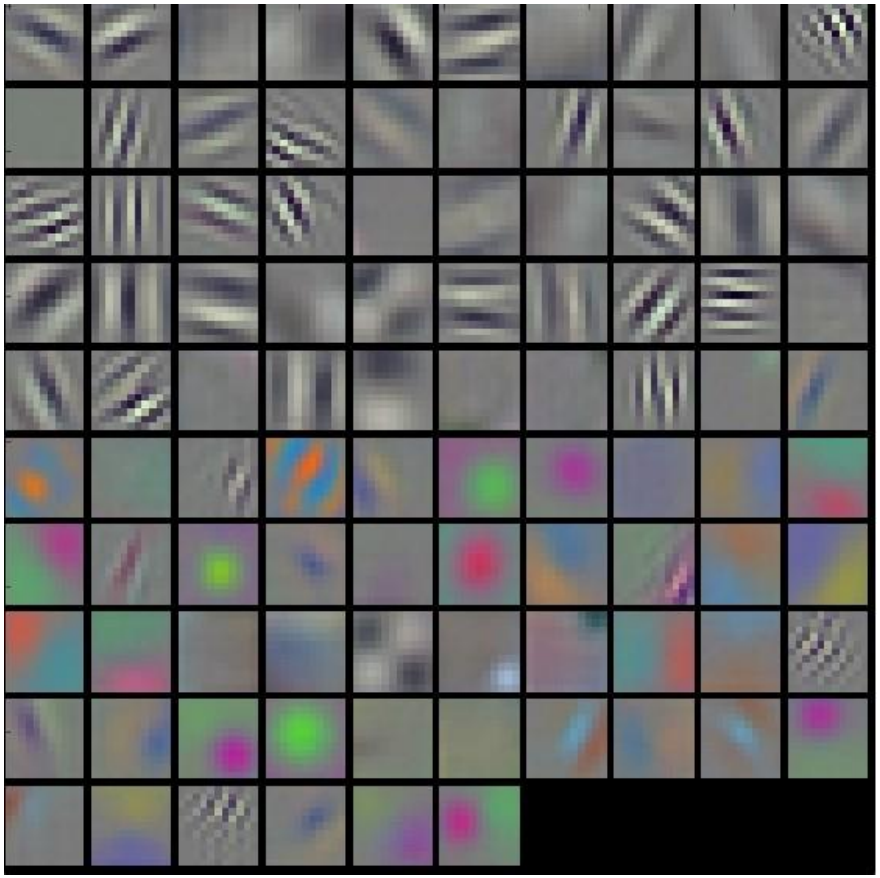
- CNNs generally difficult to train, since many architectural decisions have to be made and many hyper parameters have to be set
- *Input images*: Dimensions that are multiples of 2 (32, 64, 128, etc.)
- *Kernel size*: 3x3, more seldom 5x5; stride of 1, zero-padding
  - Exception: On insufficient GPU RAM one can choose 7x7 as first layer
- *Amount of feature maps / filters per layer*: The smaller the image, the more feature maps can be computed at the same costs; often sensible to keep costs constant
- *Pooling*: None or max-pooling with 2x2

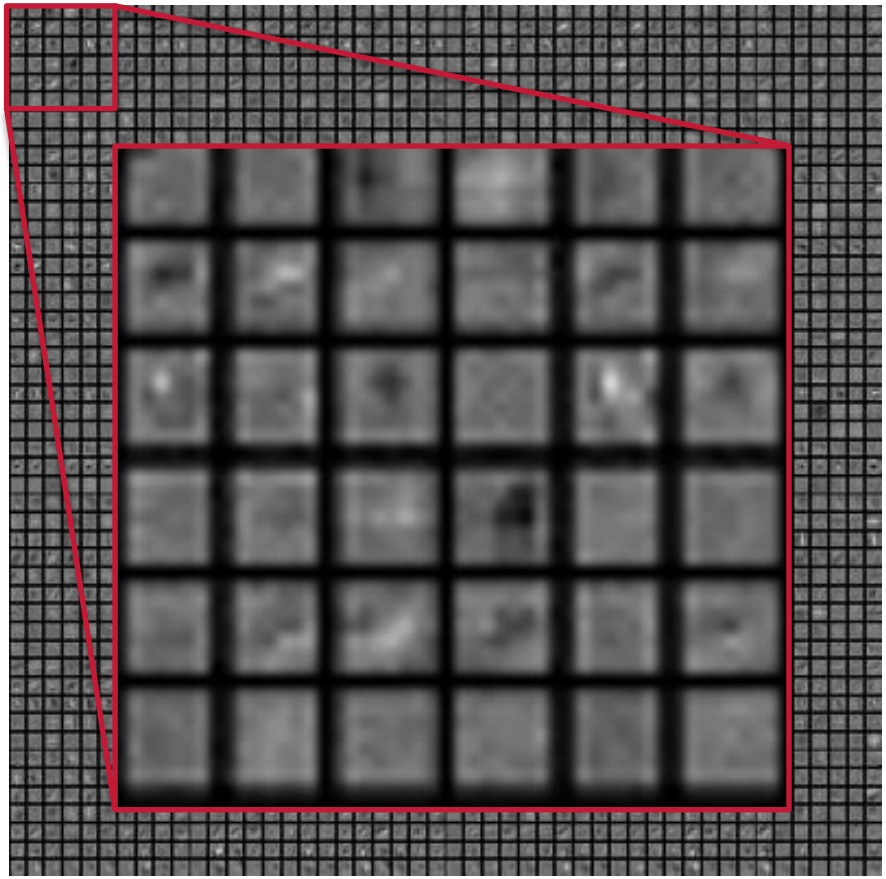
Visualization of ConvNets

How to visualize what has been learned?

1. Filters directly
2. Guided backpropagation
3. Feature Maps
4. Image creating the maximum activation at a neuron
5. Convoluted images
6. Low dimensional, pre-last layer distance representation
7. Occlusion window

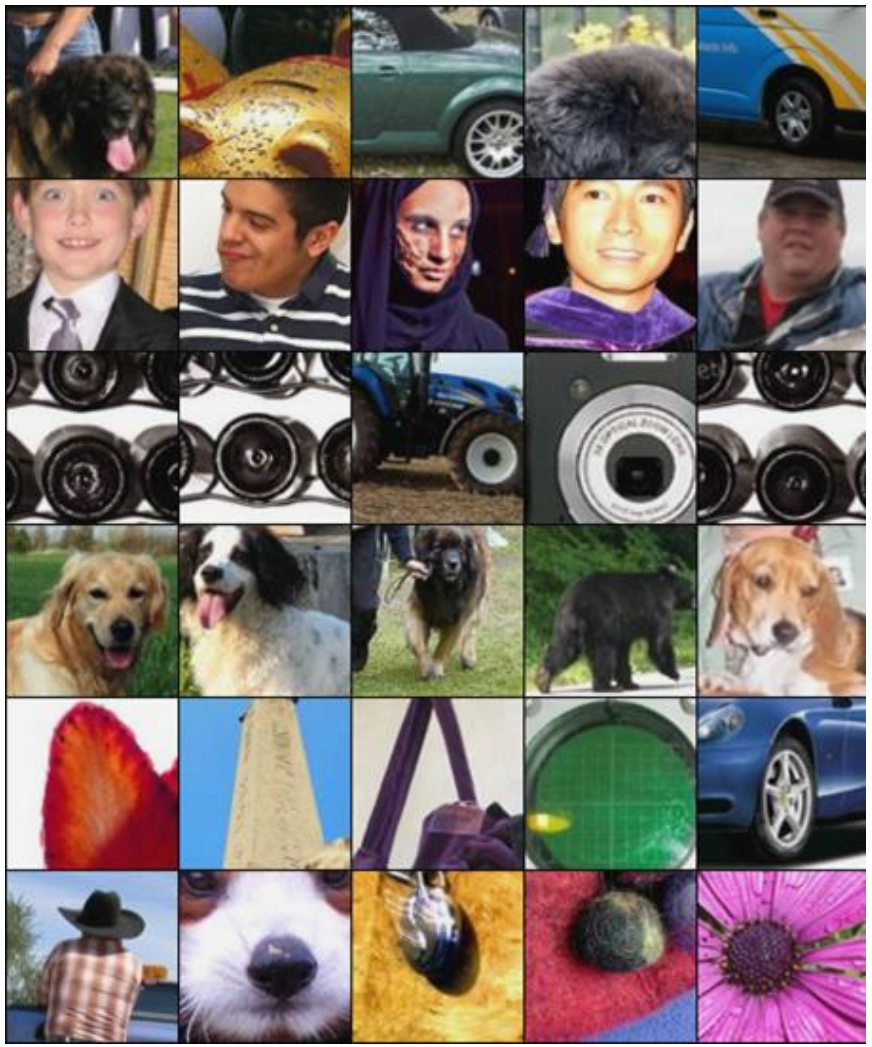
Visualization of ConvNets: Filters directly





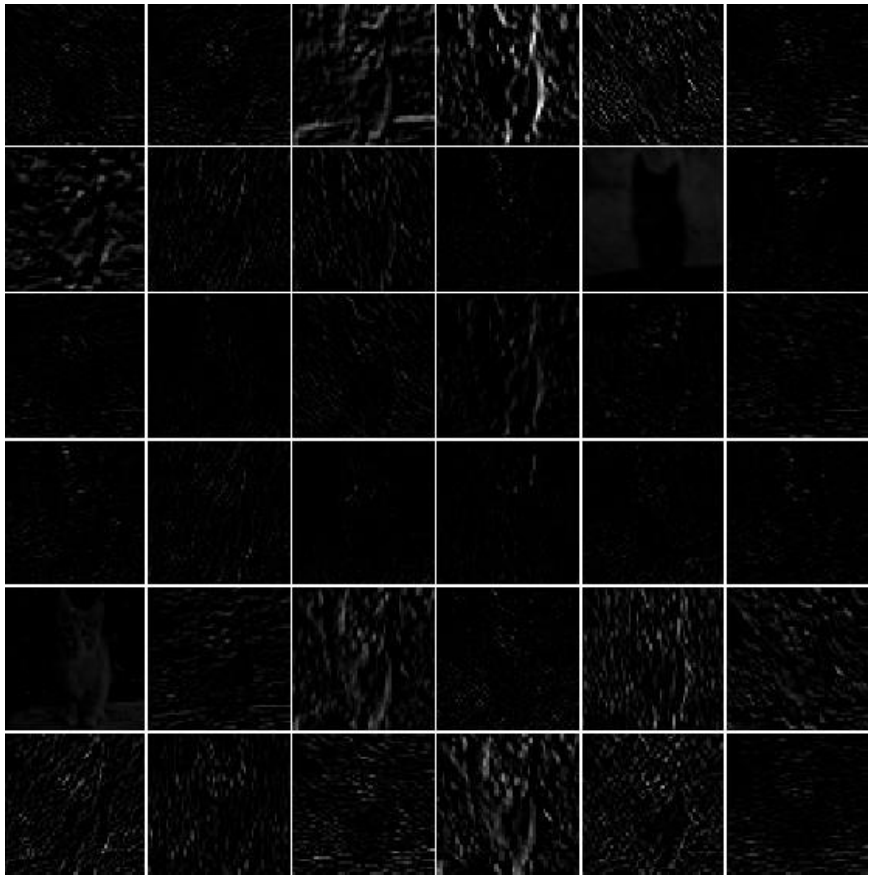
Source: http://cs231n.github.io/understanding-cnn/

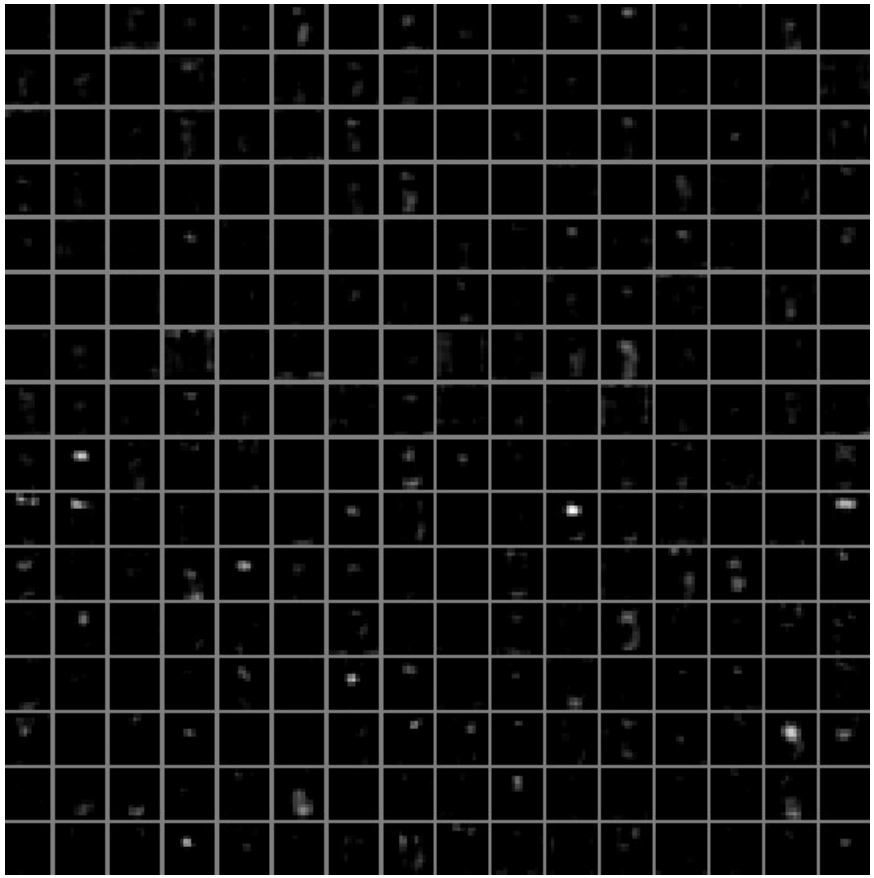
Visualization of ConvNets: Filters backprojected



Source: From Striving for Simplicity: The All Convolutional Net", http://arxiv.org/pdf/1412.6806v3.pdf

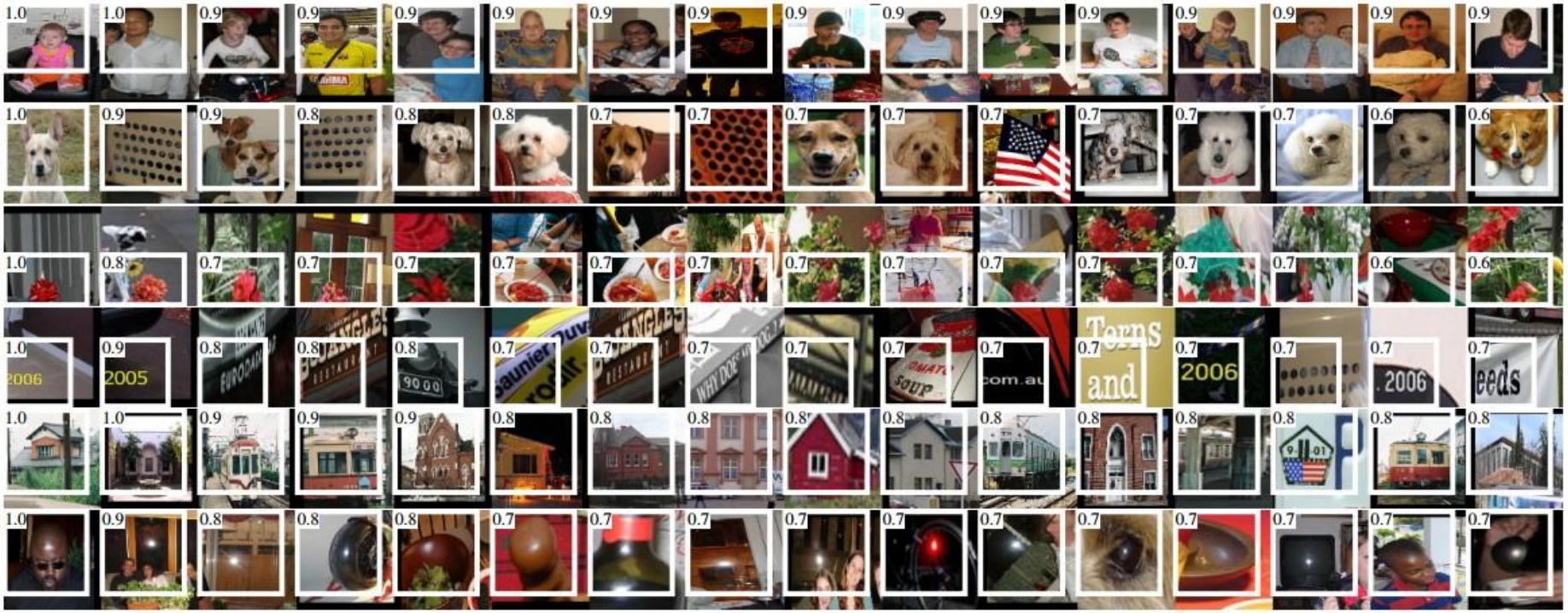
Visualization of ConvNets: Feature maps





Source: http://cs231n.github.io/understanding-cnn/

Visualization of ConvNets: Max activations

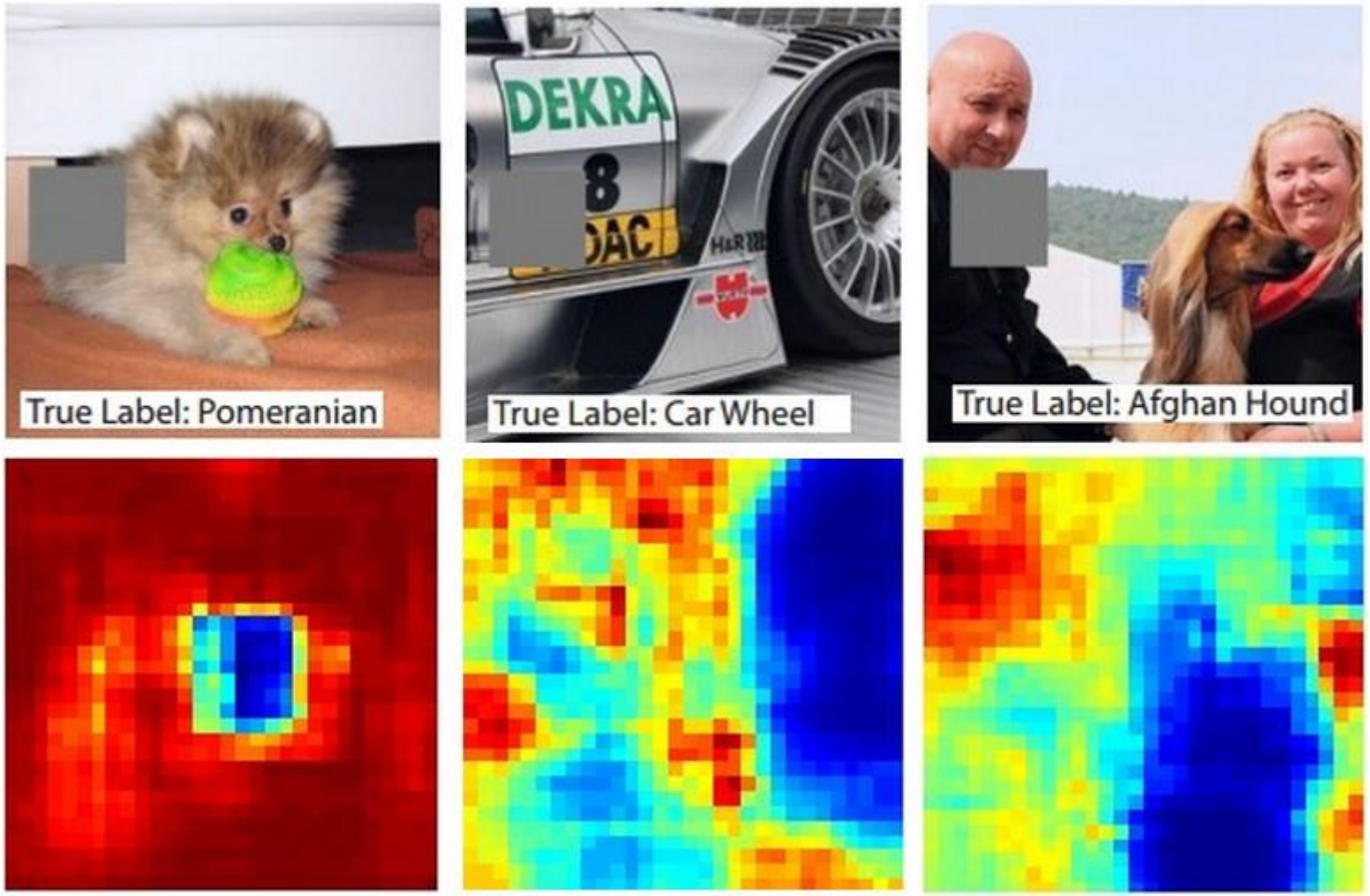


Source: Ross Girshick et al., "Rich feature hierarchies for accurate object detection and semantic segmentation"

Visualization of ConvNets: Distance

Source: http://cs.stanford.edu/people/karpathy/cnnembed/

Visualization of ConvNets: Occlusion window



Source: Matthew Zeiler's Visualizing and Understanding Convolutional Networks