Deep Learning for Incipient Slip Detection

Robert Haschke

Center of Excellence Cognitive Interaction Technology (CITEC)

Overview

- Success Stories of Deep Learning
- Motivation for Deep Architectures
- Ingredients of Deep Learning

Success Stories of Deep Learning

Success Stories of Deep Learning

- Vision (ImageNet competition)
  - 1.3 million images, 1000 classes
  - top 5 error of ~5%
    (matches human performance)

Success Stories of Deep Learning

- Vision (ImageNet competition)
  - 1.3 million images, 1000 classes
  - top 5 error of ~5%
    (matches human performance)
- Natural Language Processing (Siri, ...)

Success Stories of Deep Learning

- Vision (ImageNet competition)
  - 1.3 million images, 1000 classes
  - top 5 error of ~5%
    (matches human performance)
- Natural Language Processing (Siri, ...)
- Word Embeddings

Success Stories of Deep Learning

- Vision (ImageNet competition)
  - 1.3 million images, 1000 classes
  - top 5 error of ~5%
    (matches human performance)
- Natural Language Processing (Siri, ...)
- Word Embeddings
- Text Processing
  - Automatic Translation

ImageNet Examples

Word Embeddings for Language Processing

- represent words by vectors $\in \mathbb{R}^n$
- learned from word co-occurence in large text-corpora

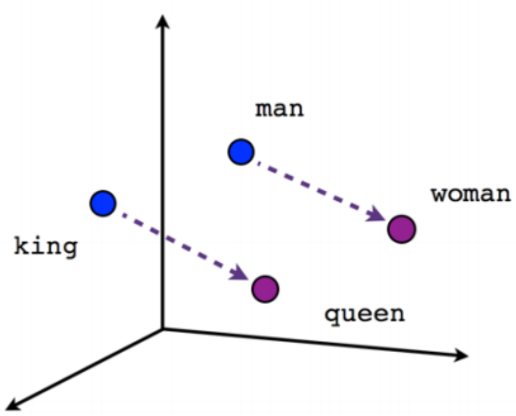
$$w_{-2}, w_{-1}, \boldsymbol{w}_{*}, w_1, w_2$$

Word Embeddings for Language Processing

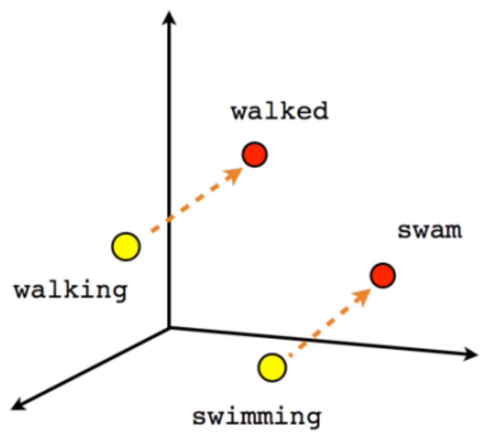
- represent words by vectors $\in \mathbb{R}^n$
- learned from word co-occurence in large text-corpora

$w_{-2}, w_{-1}, \boldsymbol{w}_{*}, w_1, w_2$

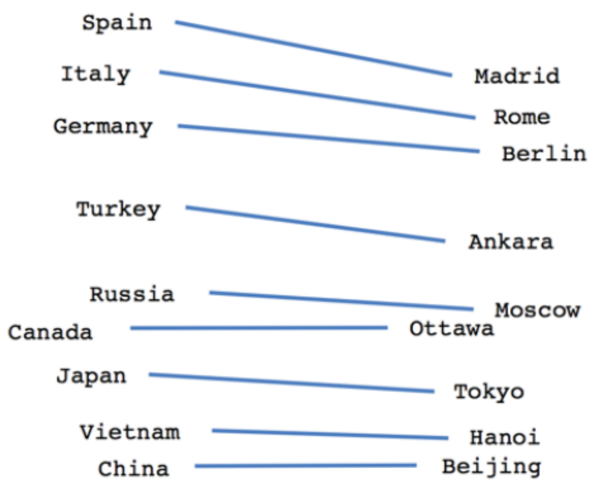
- semantics encoded in the (linear) topology of the space



Male-Female



Verb tense



Country-Capital

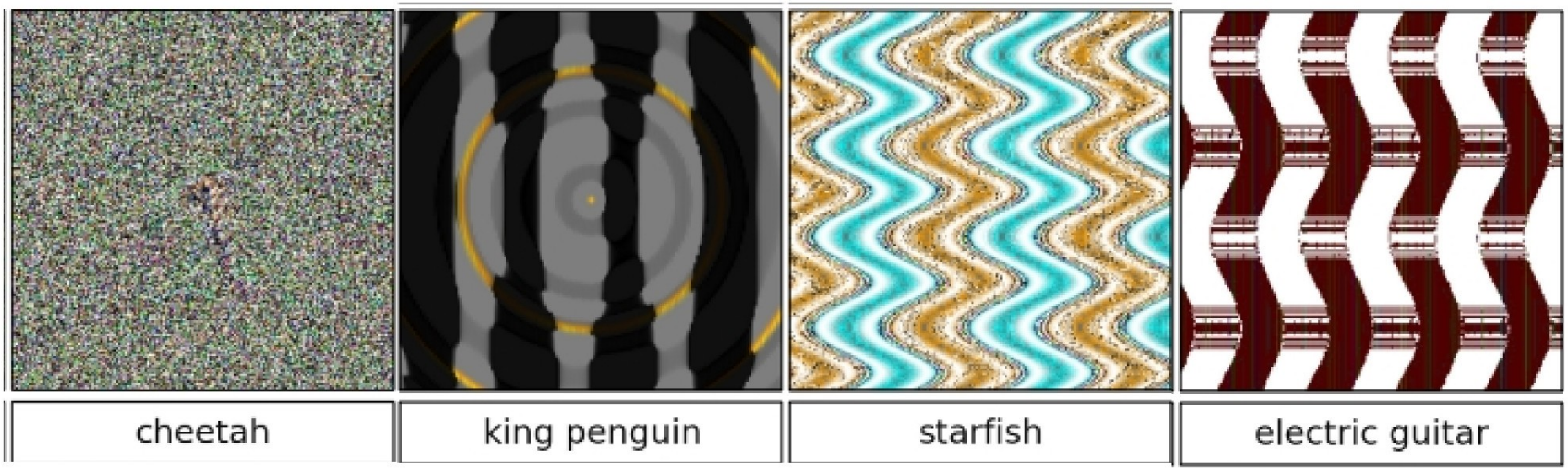
Fusing Vision and Speech

- instead of softmax layer, feed output to RNN
- RNN trained on human description of images

Vinyals et al. 2014 *Show and Tell: A Neural Image Caption Generator*

Limitations of Neural Networks

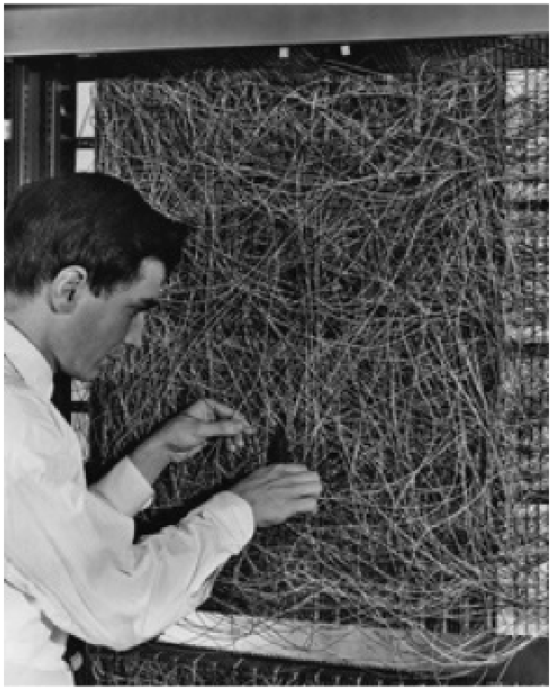
- confidence >99.6%
- generated with Genetic Algorithms



Nguyen et al. 2014 *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*

Deep Learning History

- 1958 Perceptron (Rosenblatt)
- 1980 Neocognitron (Fukushima)
- 1982 Hopfield network, SOM (Kohonen)
- 1985 Boltzmann machines (Ackley et al)
- 1986 MLP + backpropagation (Rumelhart)
- 1988 RBF networks (Broomhead + Lowe)
- 1989 Autoencoders (Baldi + Hornik)
- 1989 Convolutional Network (LeCun)
- 1993 Sparse Coding (Field)
- 2000s Sparse, Probabilistic, and Layer-wise models (Hinton, Bengio, Ng)
- 2012 DL clearly won ImageNet competition (Krizhevsky et al.)



Rosenblatt's Perceptron

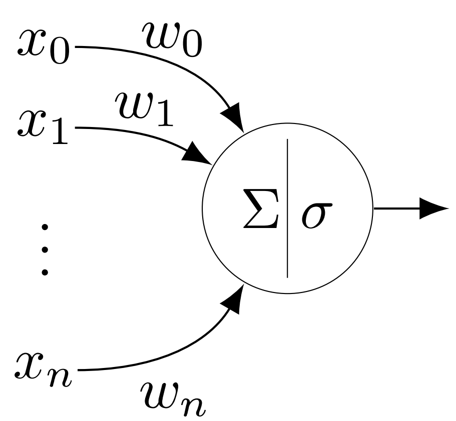
Why Now?

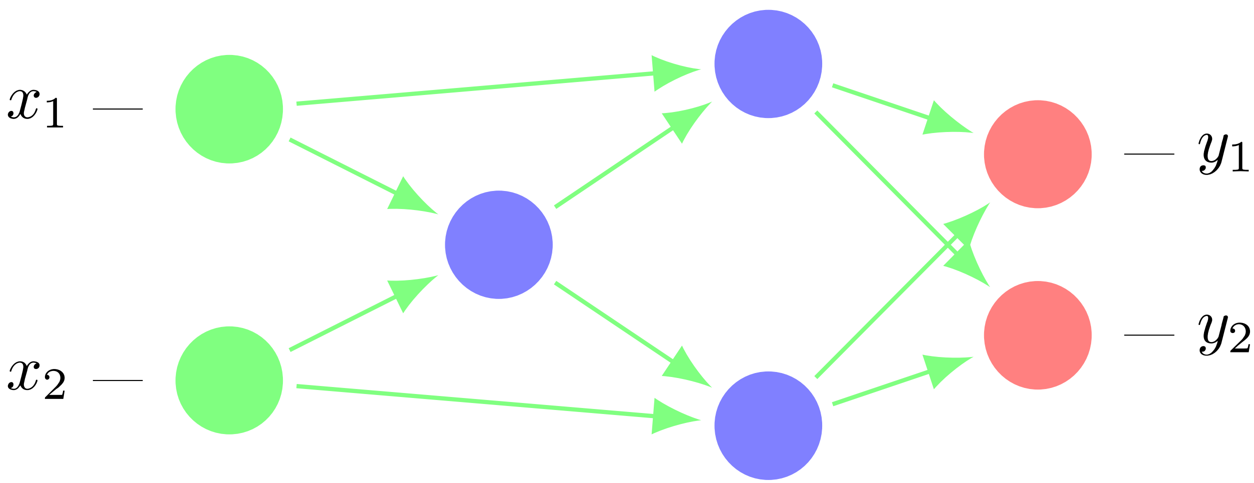
- Big Data
  - ImageNet et al: millions of labeled images (crowd-sourced)
- Computing Power – GPUs
  - terabytes/s memory bandwidth
  - teraflops compute
- Improved Methods
  - efficient + numerically robust learning frameworks
  - new optimization methods

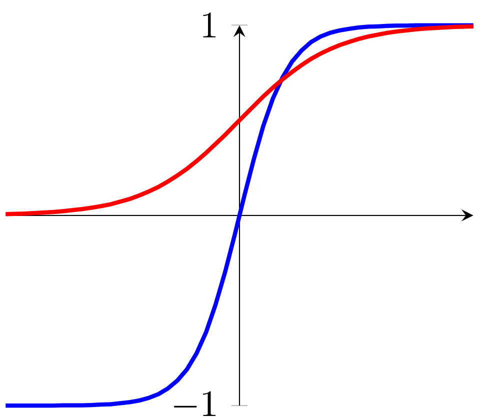
How are these amazing results achieved?

Neural Networks

- simple units layered in a network structure
- weighted sum of inputs: $\mathrm{h} = \sum w_i x_i$
- nonlinear activation: $\mathrm{y} = \sigma(h)$

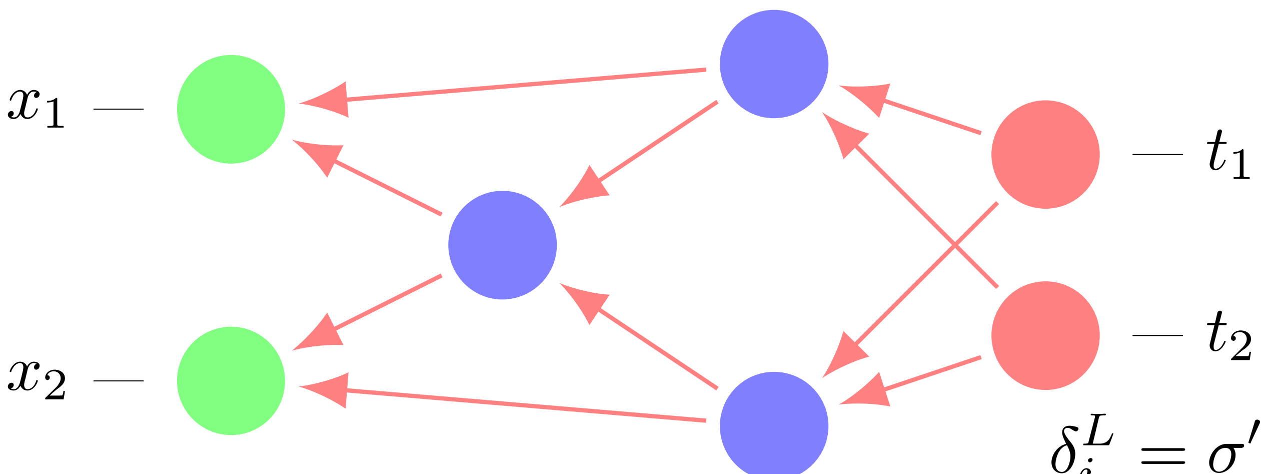






Neural Network Learning

- learning by backpropagation of errors δ_i^ν
- layered structure + chain rule = backpropagation

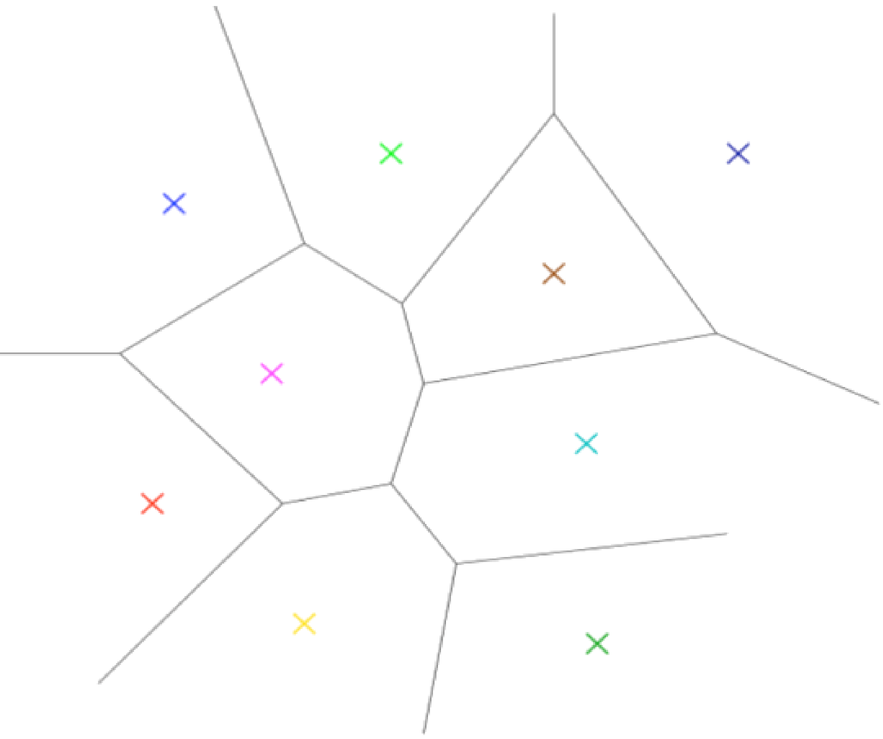


$$\delta_i^L = \sigma'(h_i^L) \cdot (y_i^L - t_i^\alpha)$$

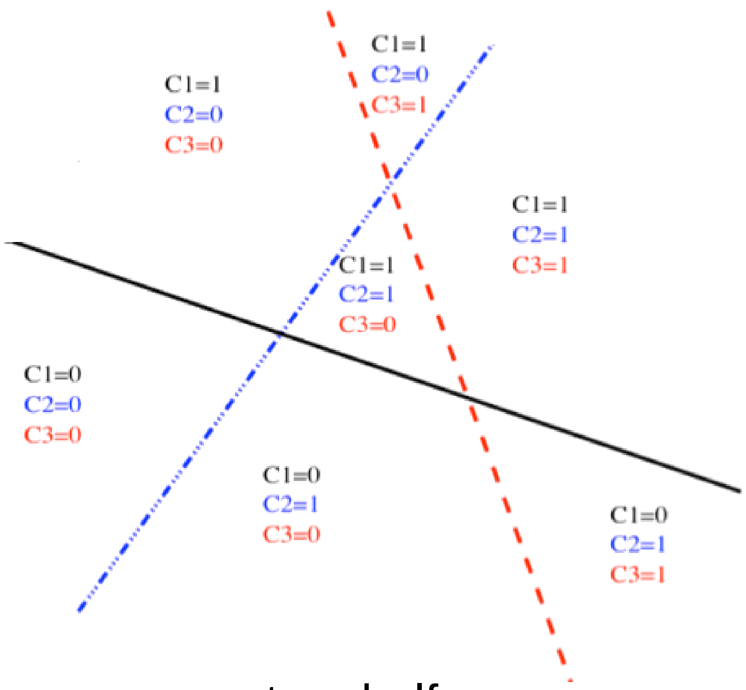
$$\delta_i^\nu = \sigma'(h_i^\nu) \sum_k \omega_{k,i}^{\nu+1,\nu} \delta_k^{\nu+1}$$

Distributed Representation

- prototype-based representation needs many examples



prototype-based learning



perceptron half-spaces

Distributed Representation

- prototype-based representation needs many examples
- *composition* of features is exponentially more efficient

Distributed Representation

- prototype-based representation needs many examples
- *composition* of features is exponentially more efficient

  Consider a network whose hidden units represent the features:
  - person is male / female
  - person is young / old
  - person wears glasses
  - person has beard

Distributed Representation

- prototype-based representation needs many examples
- *composition* of features is exponentially more efficient

  Consider a network whose hidden units represent the features:
  - person is male / female
  - person is young / old
  - person wears glasses
  - person has beard

  Given n features and each feature requires $O(k)$ parameters, need $O(nk)$ examples.

  Prototype-based methods would require $O(k^n)$ examples.

Distributed Representation

- prototype-based representation needs many examples
- *composition* of features is exponentially more efficient
- prior assumption:
  compositionality is useful to describe real-world
- exploit underlying structure of the world

Backpropagation Doesn't Scale to Deep Nets

Deep nets perform *worse* than shallow nets
when trained with randomly-initialized backpropagation.

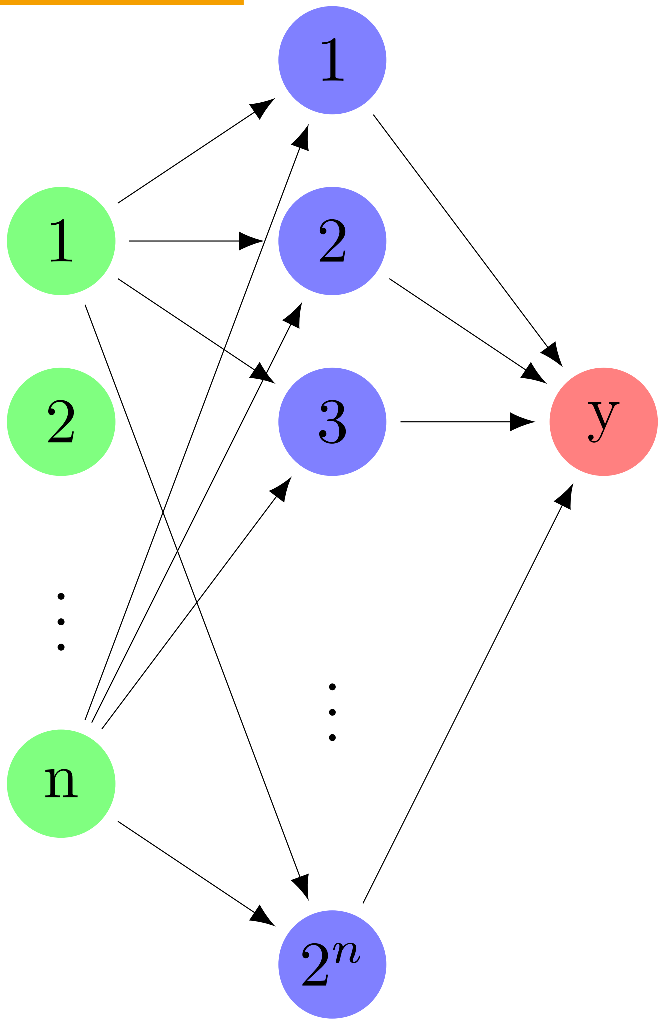
	training	validation	test
shallow net random initialization	0.004%	1.8%	1.9%
deep net random initialization	0.004%	2.1%	2.4%
deep net unsupervised pre-training	0%	1.4%	1.4%

Bengio et al., NIPS 2007

Why going deep?

- one hidden layer of
  - neurons
  - RBF units
  - logic units

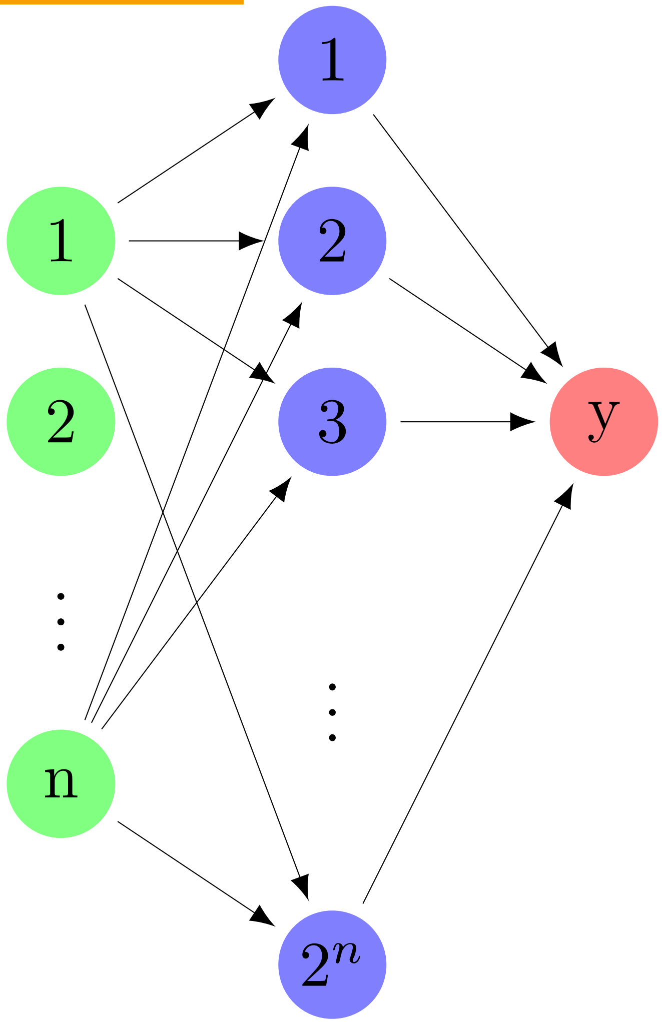
  is a *universal approximator*



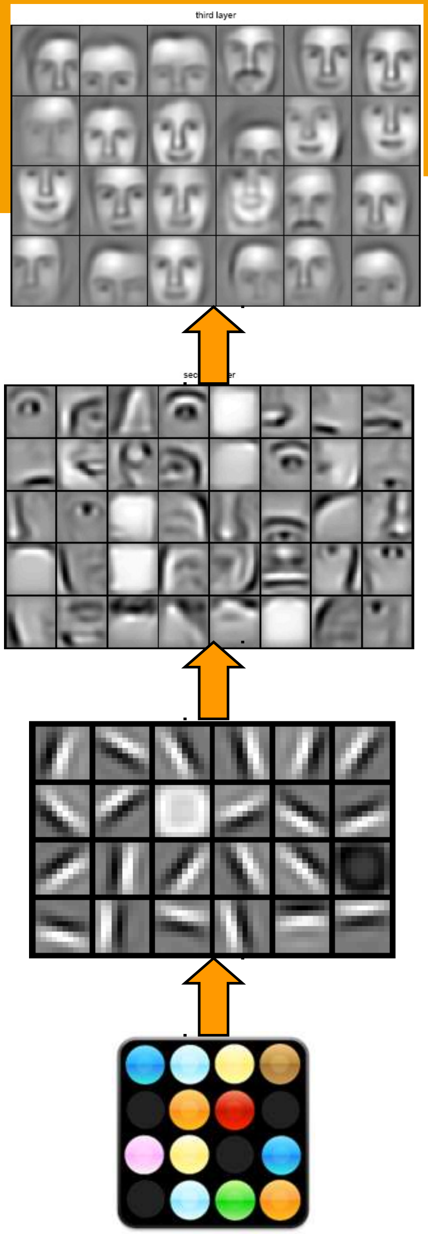
Why going deep?

- one hidden layer of
  - neurons
  - RBF units
  - logic units

  is a *universal approximator*
- stacking multiple hidden layers
  is more efficient than a single one
  Montufar et al, NIPS 2014



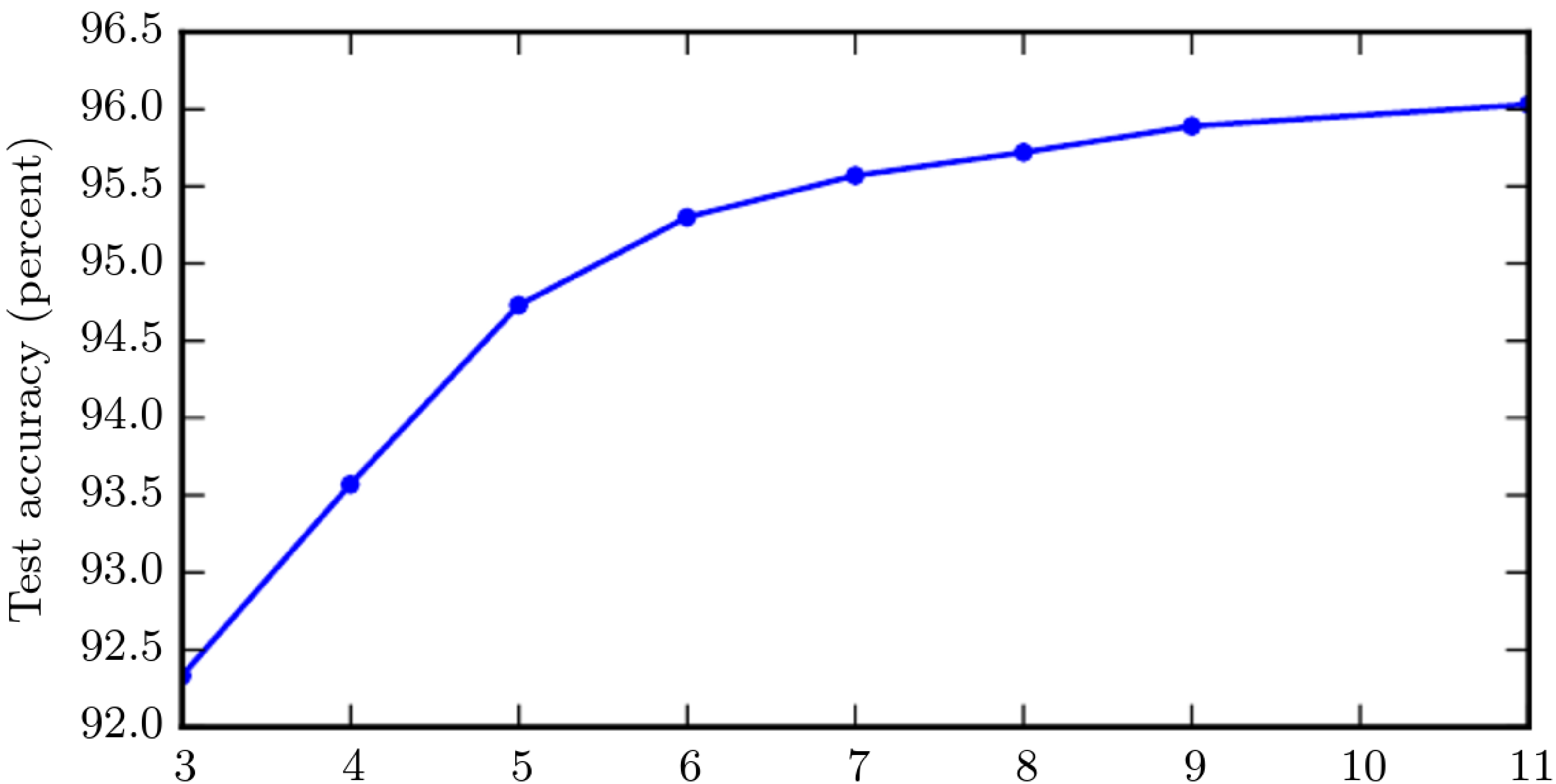
Why going deep?



- one hidden layer of
  - neurons
  - RBF units
  - logic units

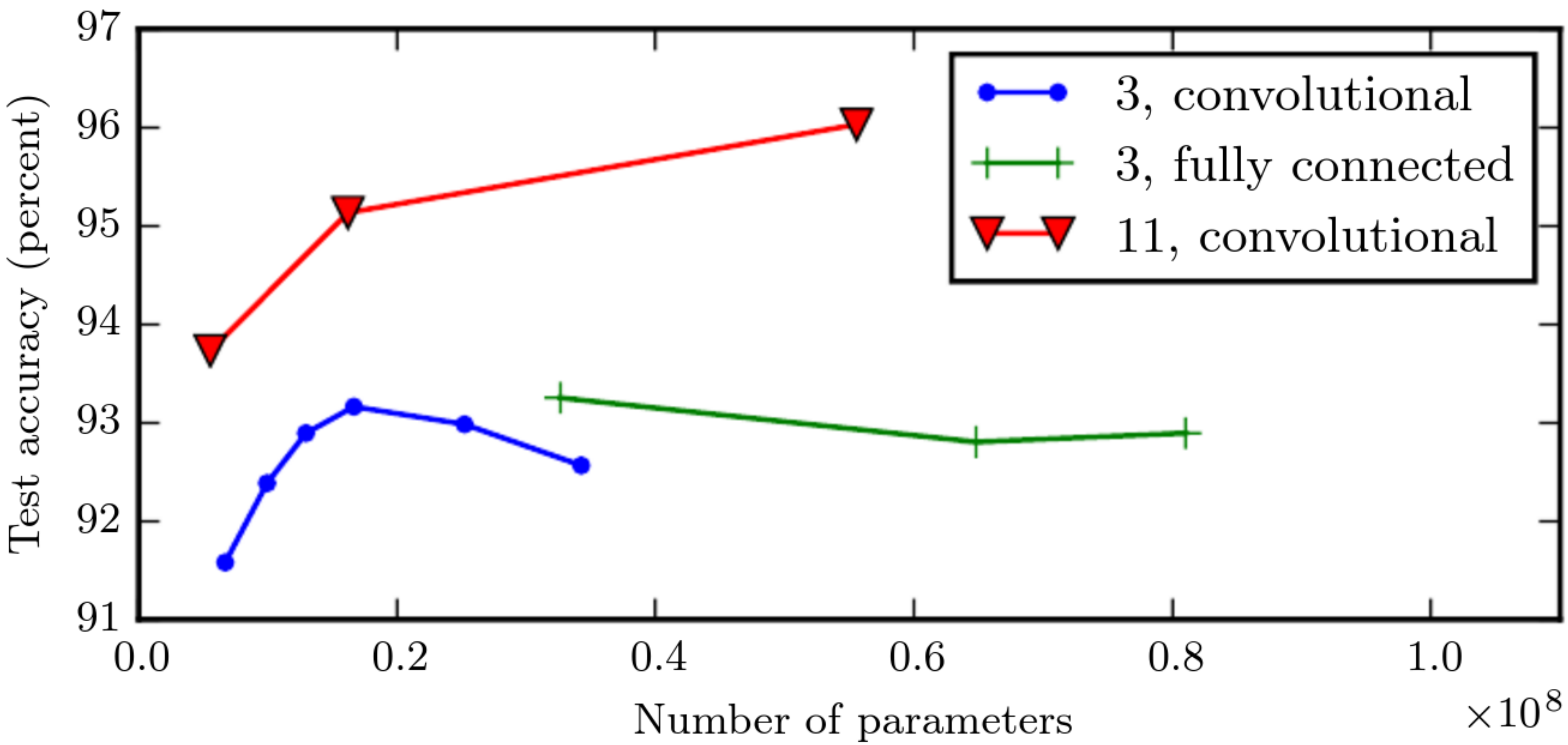
  is a *universal approximator*
- stacking multiple hidden layers
  is more efficient than a single one
  Montufar et al, NIPS 2014
- hierarchy allows for
  more complex features

Recognizing numbers (Google Street View)



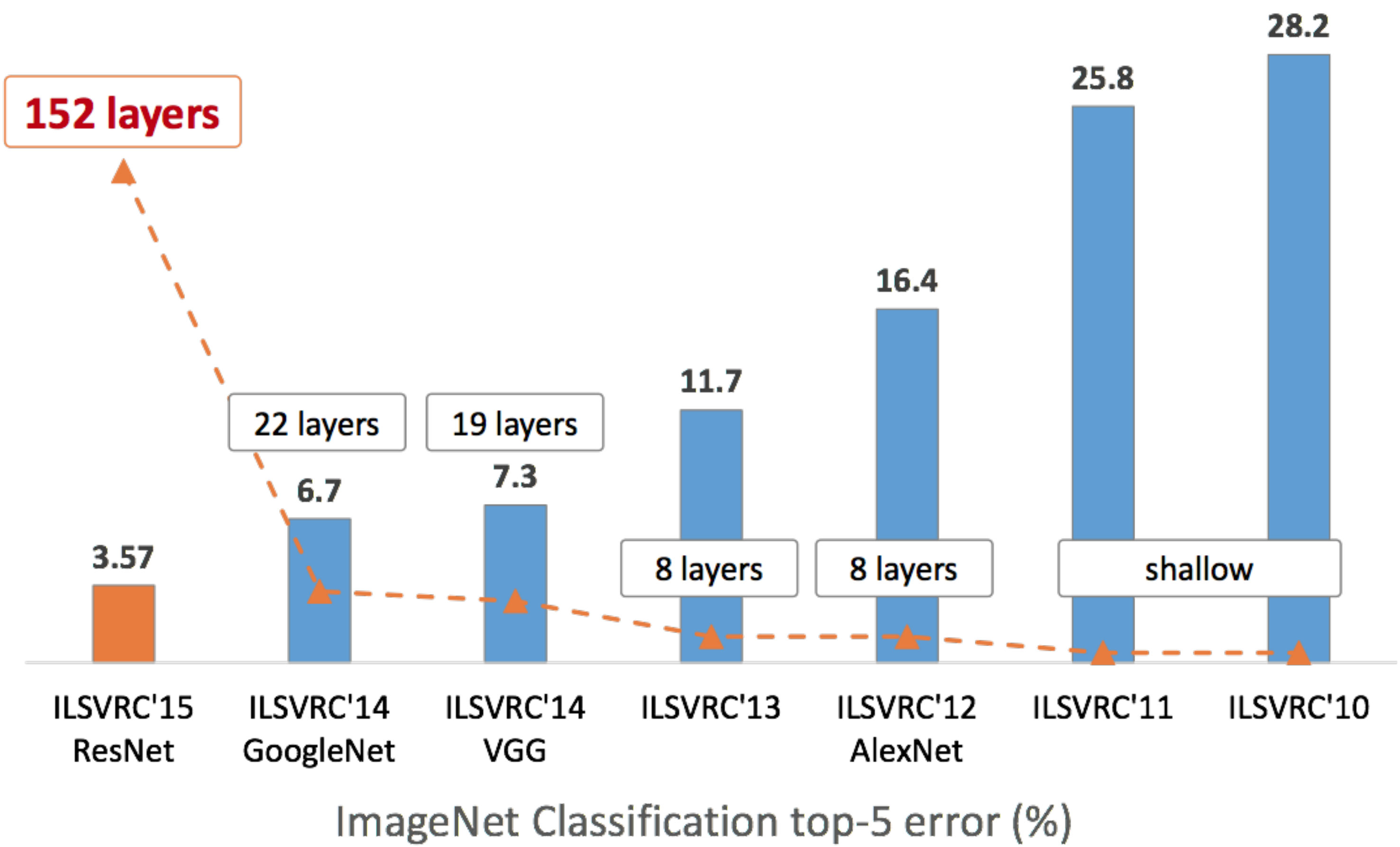
[graph credit Goodfellow, 2014]

Deep models make better use of more params



[graph credit Goodfellow, 2014]

Increase of Depth in ImageNet Classification



- ☐ dog
- ☑ car
- ☑ horse
- ☐ bike
- ☐ cat
- ☐ bottle
- ☑ person

[graph credit K. He]

Hierarchy of ML

- Neural Nets learn features
- Deep Learning learns a hierarchy of features

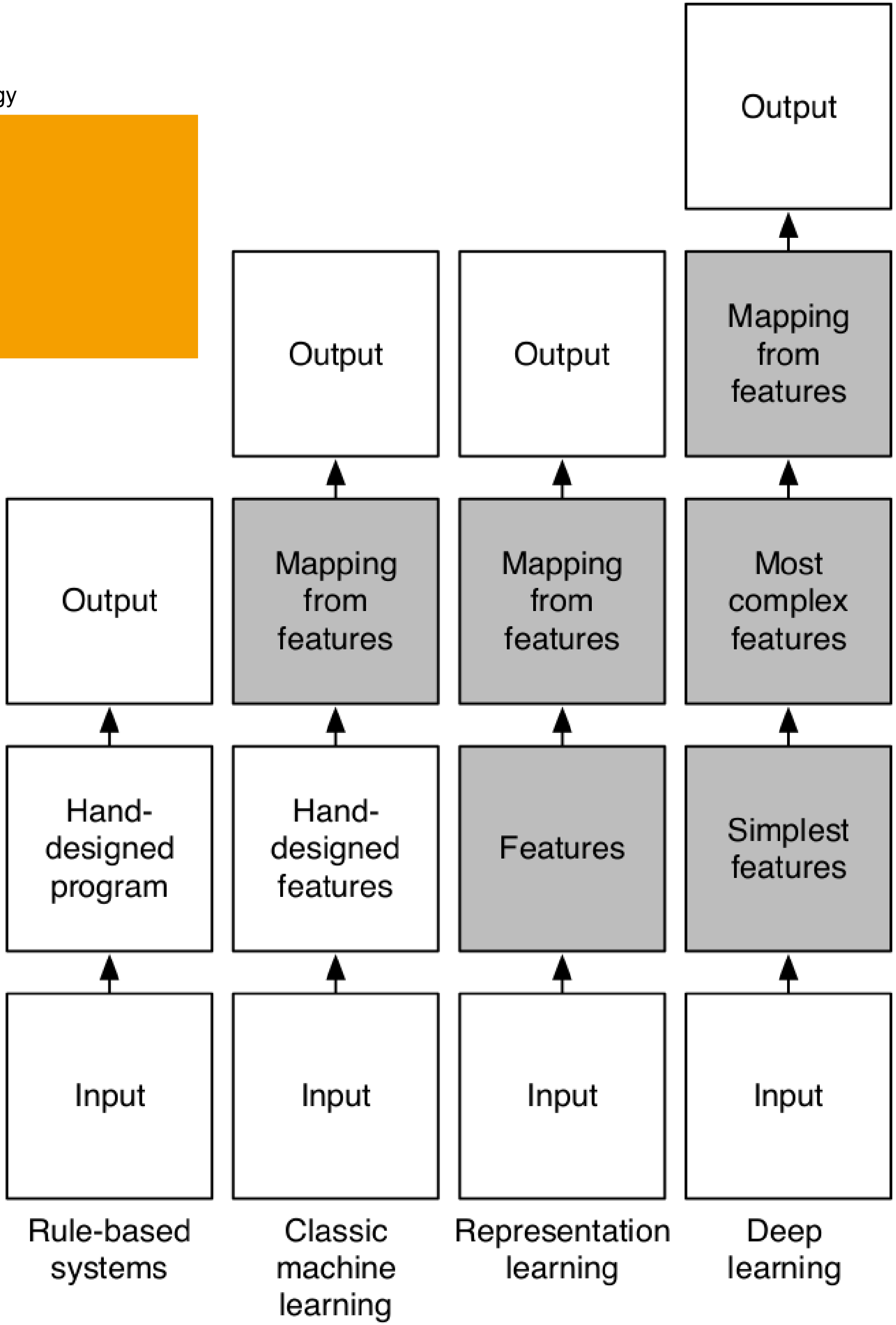


Fig. I Goodfellow

Issues with Backpropagation

- vanishing gradient
  gradient is diluted from layer to layer due to factor $\sigma' < 0$
- learning gets stuck
  especially if started far from good regions
  (random initialization)
- huge number of parameters (connection weights)

Ingredients for Successful Deep Learning

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)

Ingredients for Successful Deep Learning

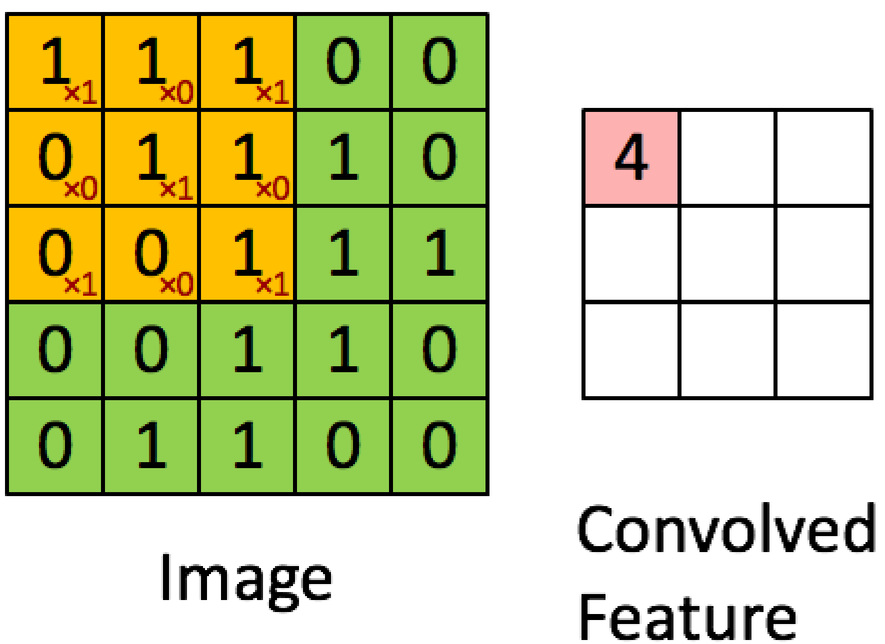
- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)
- Big Data

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)
- Big Data

Convolutional Networks

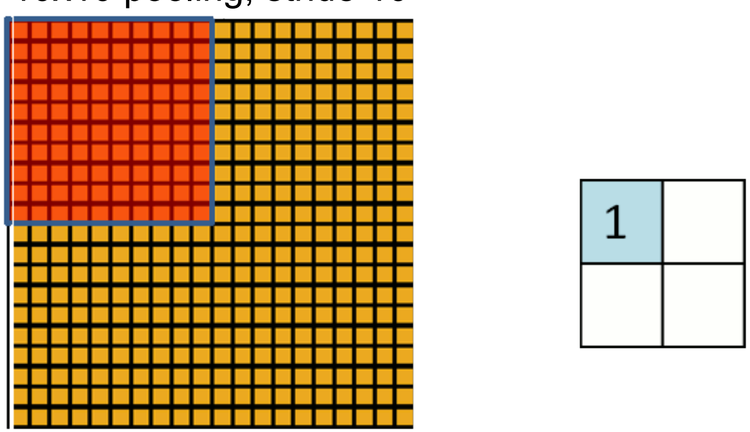
- features in *natural images* are translation-invariant features useful in one region are useful anywhere else
- motivates use of filter-bank of convolutions

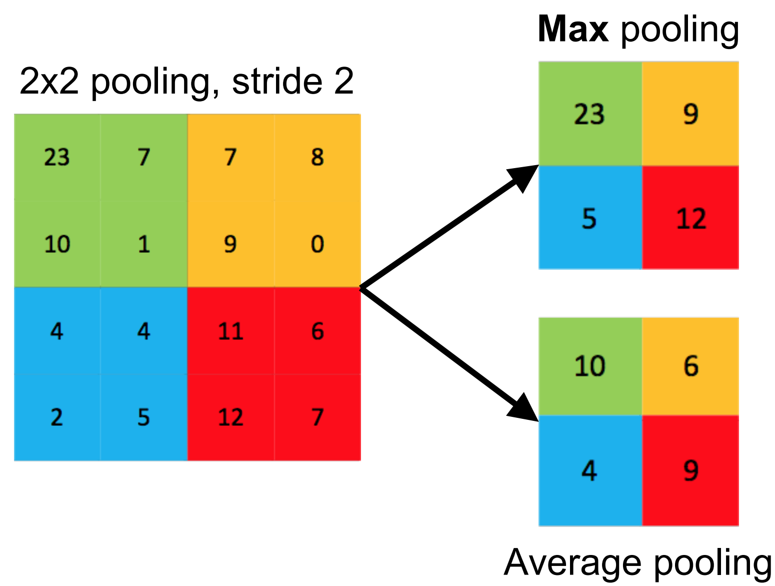


Convolutional Networks

- features in natural images are translation-invariant
  features useful in one region are useful anywhere else
- motivates use of filter-bank of convolutions
- pooling: aggregate (similar) results over an image region

10x10 pooling, stride 10



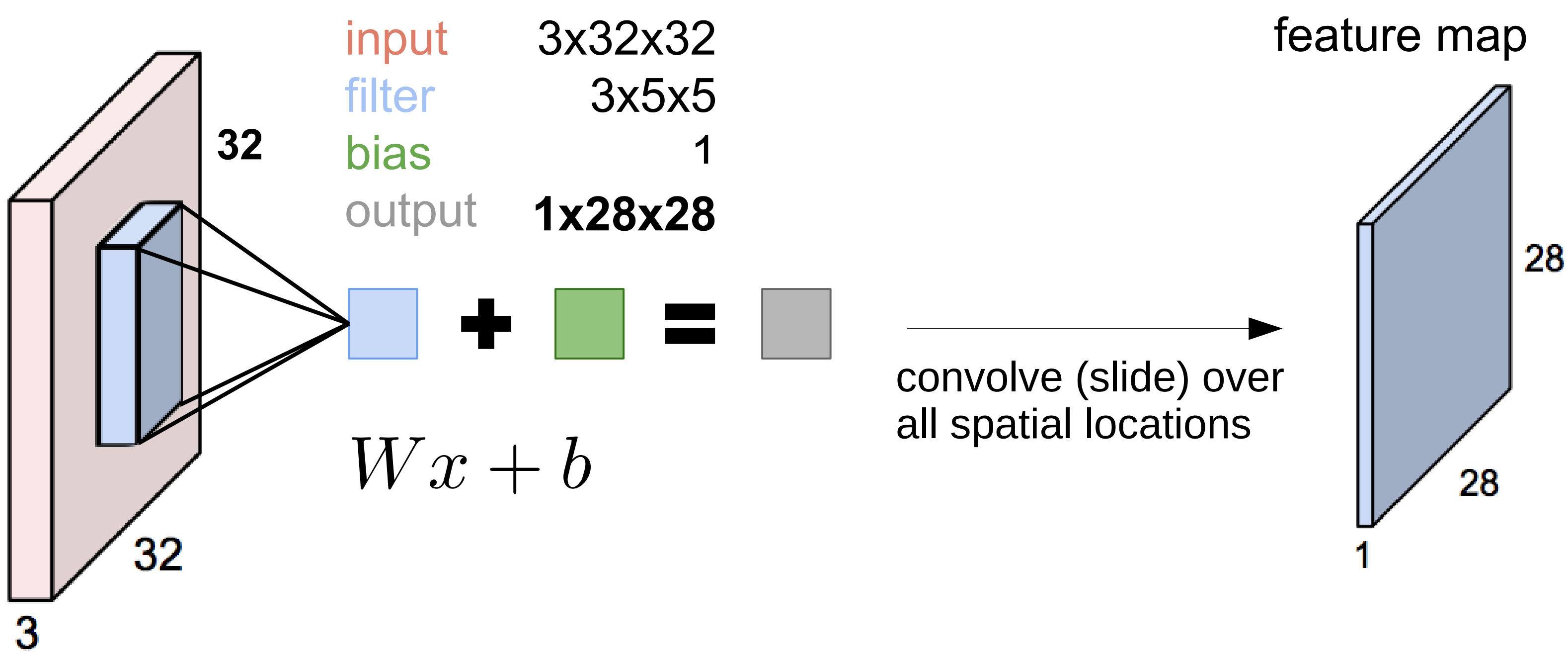


Convolutional Networks

- features in natural images are translation-invariant
  features useful in one region are useful anywhere else
- motivates use of filter-bank of convolutions
  - small filter-kernel
  - re-use filter-kernel (weight sharing)
  - dramatic reduction of weights
- pooling: aggregate (similar) results over an image region
  - reduce dimensionality of representation
  - operations: mean, max, median, …
  - overlapping or non-overlapping (stride vs. window size)

Convolution

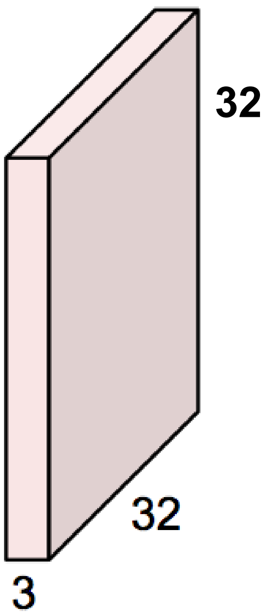
Convolving the filter with the input gives a feature map.



[figure adapted from A. Karpathy]

Convolution

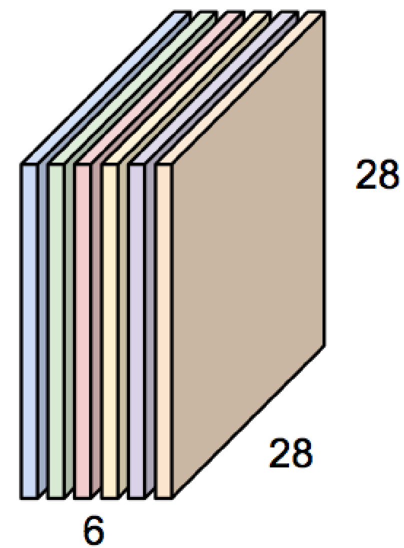
Convolving the filter with the input gives a feature map.



input	3x32x32
filter	**6**x3x5x5
bias	**6**
output	**6**x28x28

Convolution Layer
computes multiple feature maps

feature maps

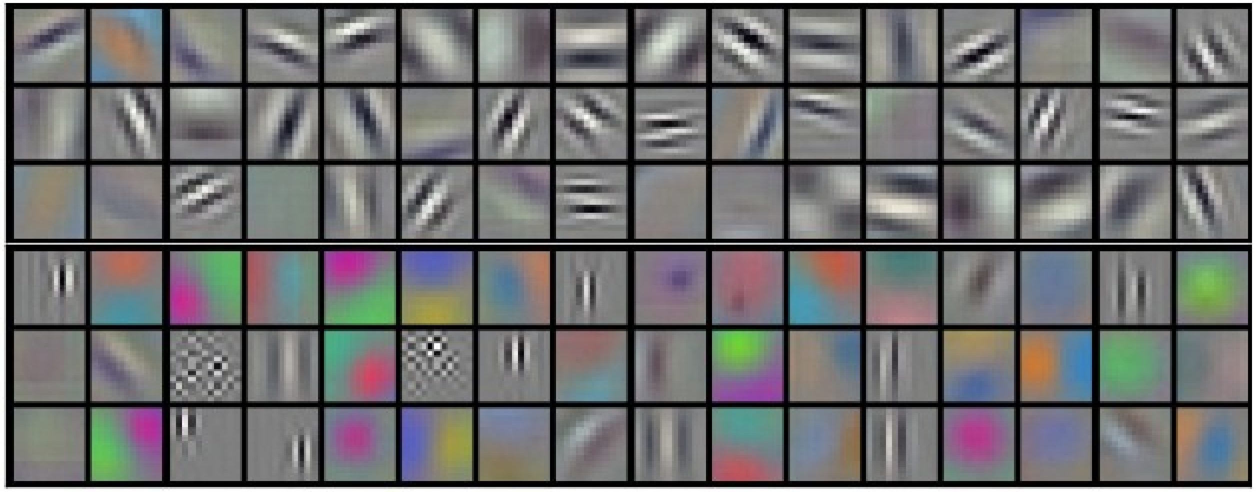


filter parameters:	**6** * $3*5^2$	= **450**
fully-conn. Params:	$3*32^2$ * **6***28^2	= **14M**

[figure adapted from A. Karpathy]

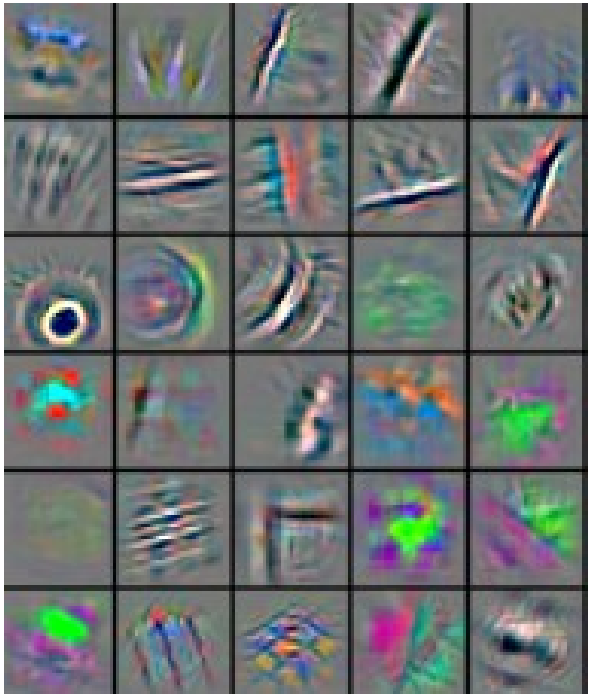
Convolution Filters provide Rich Feature Maps

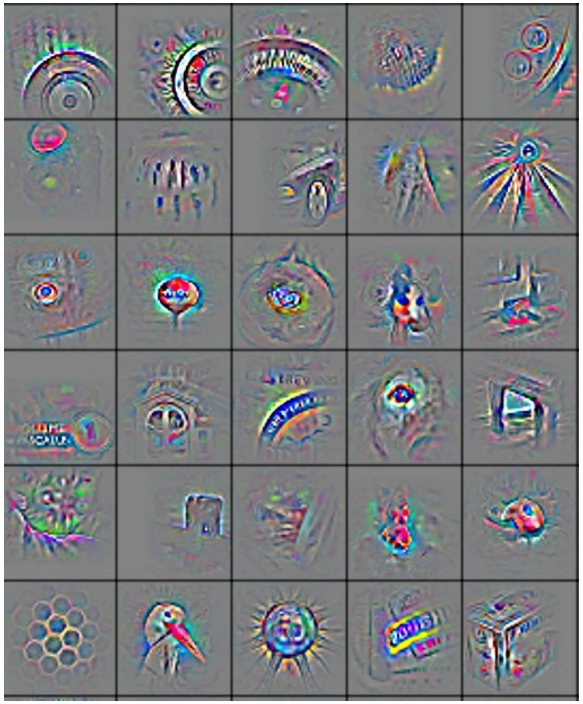
- 1st layer filters learned by AlexNet (ILSVRC‘12)
  - 96 filters of size 11x11x3
  - filters for oriented + colored edges
  - resembles Gabor filters



Convolution Filters provide Rich Feature Maps

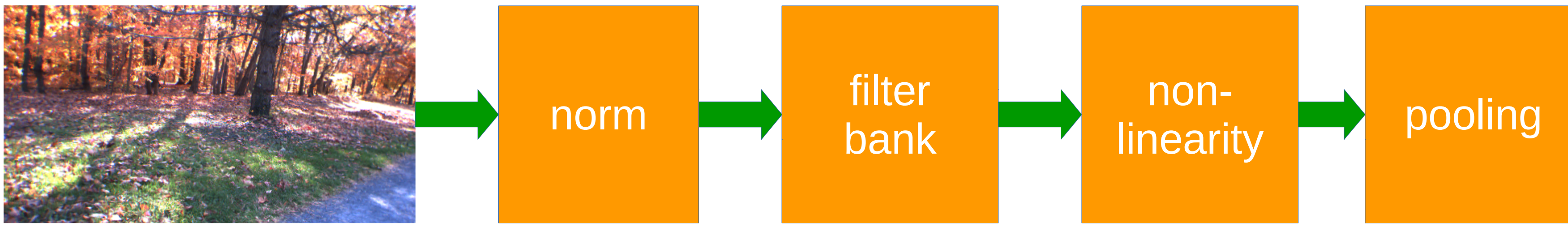
- Filters learned by Zeiler+Fergus (ILSVRC'13)
- deeper layers exhibit more complex features





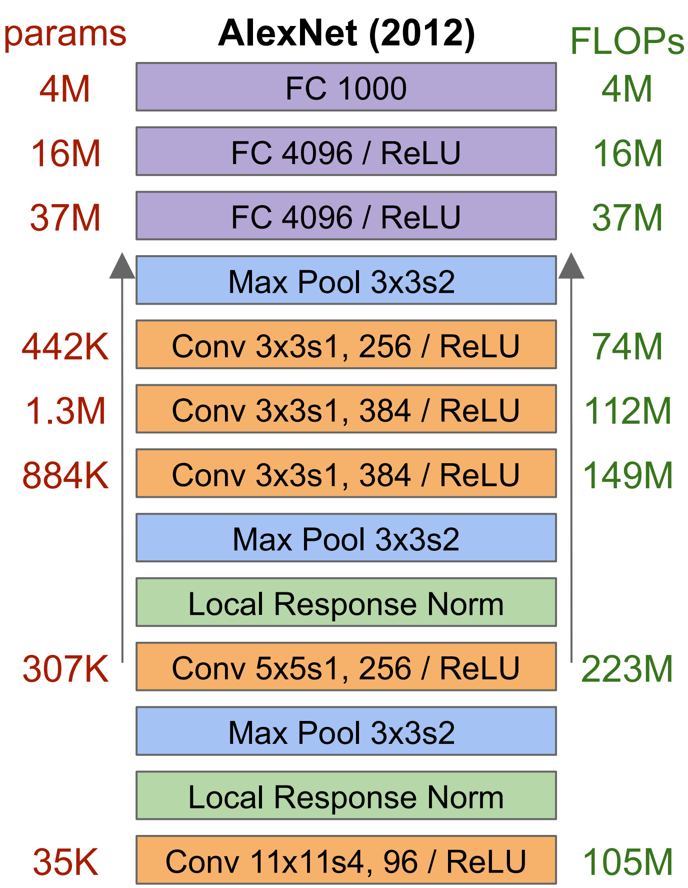
Convolutional Networks: Ingredients

- exploit *spatial structure* in input



- Normalization: average removal, variance normalization
- Filter bank: projection on overcomplete feature basis
- Non-Linearity: sparsification, saturation, lateral inhibition
- Pooling: aggregation over space or feature type
- deep convolutional networks: stack convolutional layers

Convnet Computation: 2012 & 2014



AlexNet (ILSVRC12)

- 3x227x227 input image
- **60M** parameters
- **725 M**FLOPS
- **< 1ms / image** on Titan X

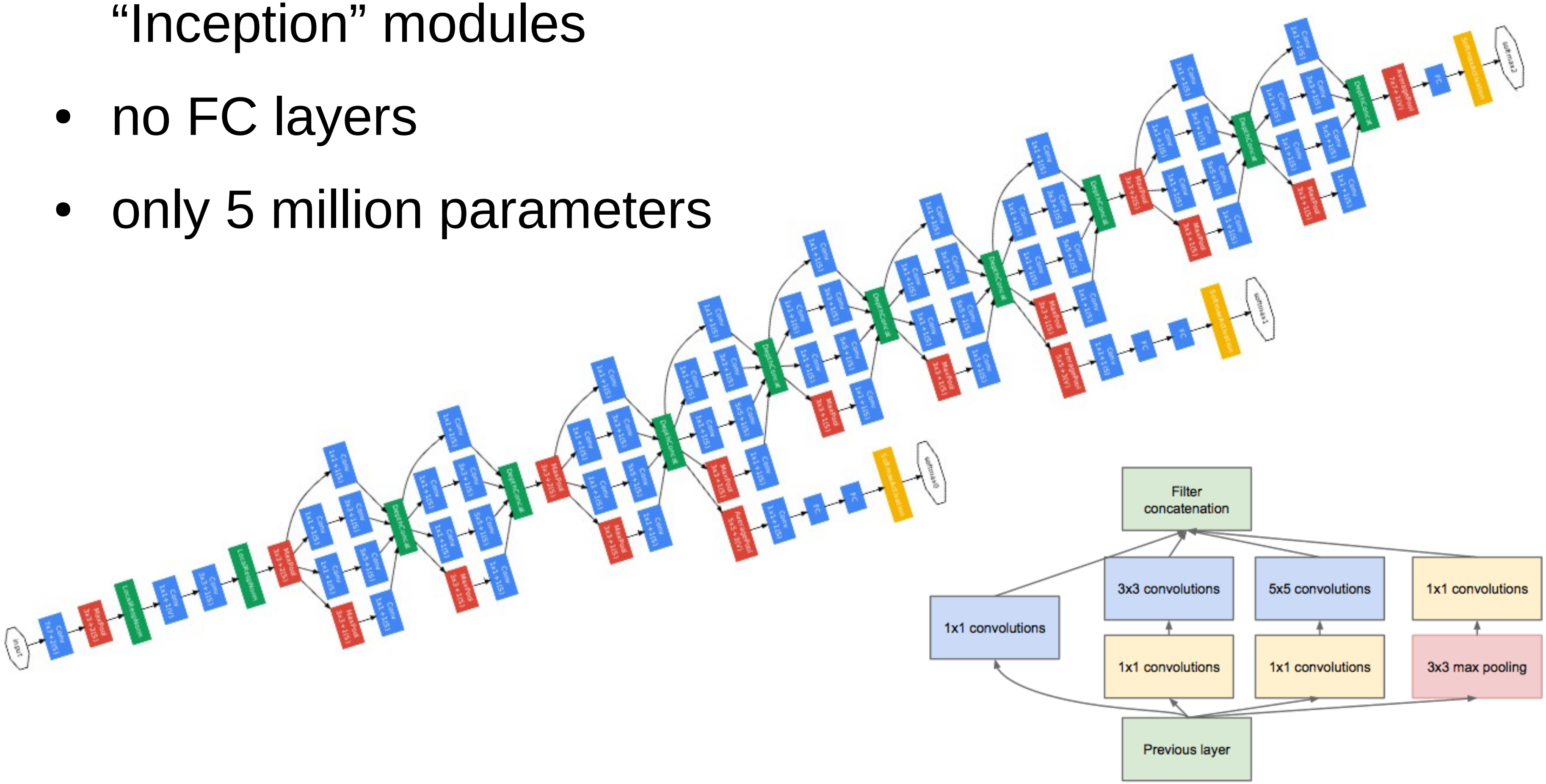
GoogleNet (ILSVRC14)

- **1.4 G**FLOPs **(200%)**
- **5M** parameters **(10%)**
- **14%** more accurate

Architecture matters!
Computational primitives are the same.

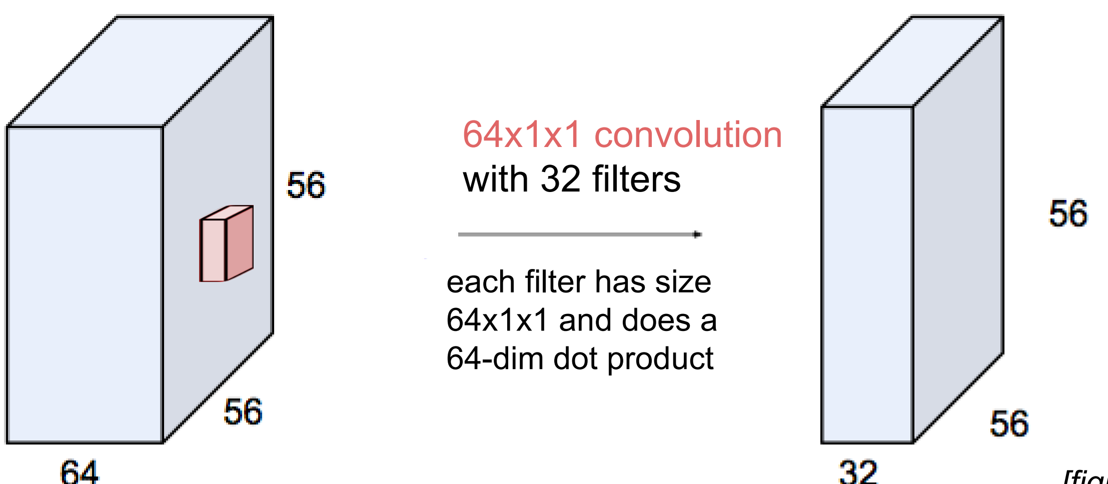
GoogLeNet (2014)

- composition of multi-scale dimension-reduced "Inception" modules
- no FC layers
- only 5 million parameters



1x1 Convolution

- compute pixel-specific combination of layer activities
- reduce channel dimension
- stack with non-linearity for deeper net
- found in many of the latest nets



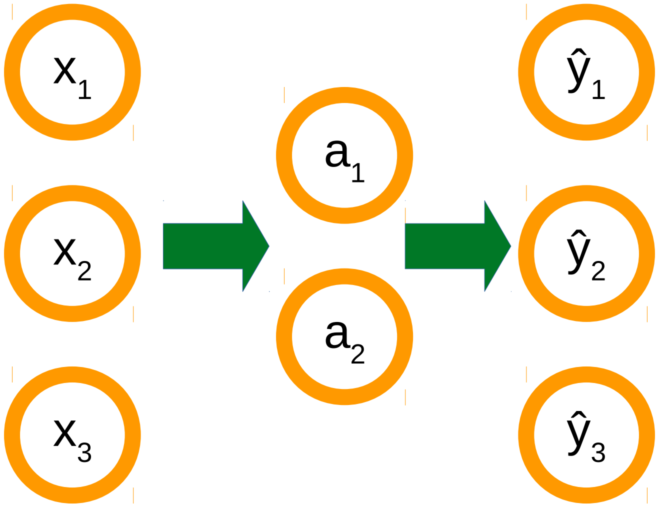
[figure credit A. Karpathy]

Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- **layer-wise training**
- boosting gradient descent
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)
- Big Data

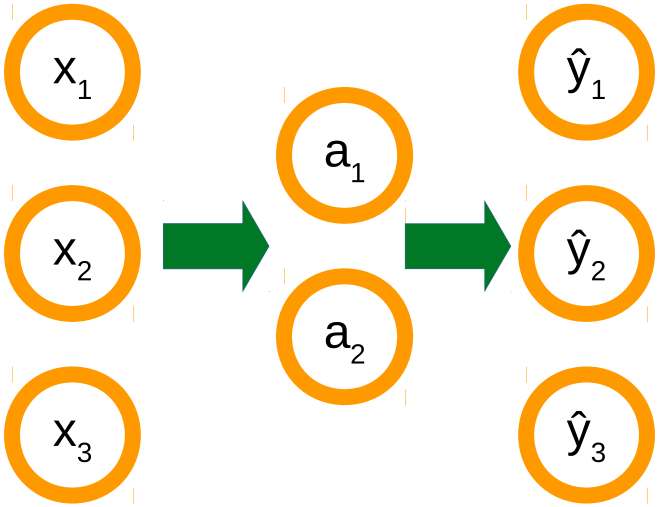
Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder

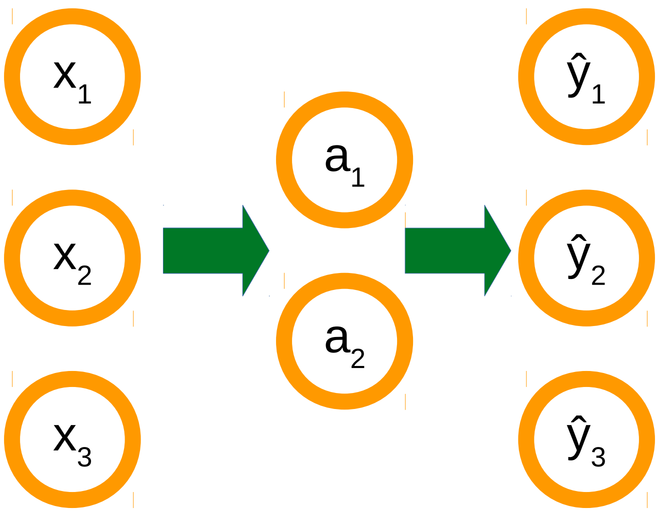


- network trained to predict input
- $\hat{y}(x) \approx x$

$$\min_{\theta} \sum_{\alpha} \|\hat{y} - x^{\alpha}\|^2 + \lambda \sum |a_i|$$

Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder

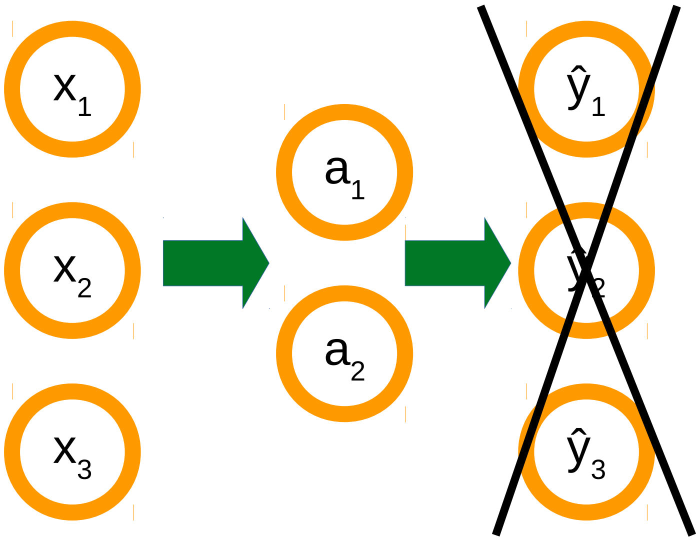


$$\min_{\theta} \sum_{\alpha} \|\hat{y} - x^{\alpha}\|^2 + \lambda \sum |a_i|$$

- network trained to predict input
- $\hat{y}(x) \approx x$
- trivial solution unless:
  - constrain #hidden units
  - constrain sparsity of hidden units

Layer-wise training: AutoEncoder

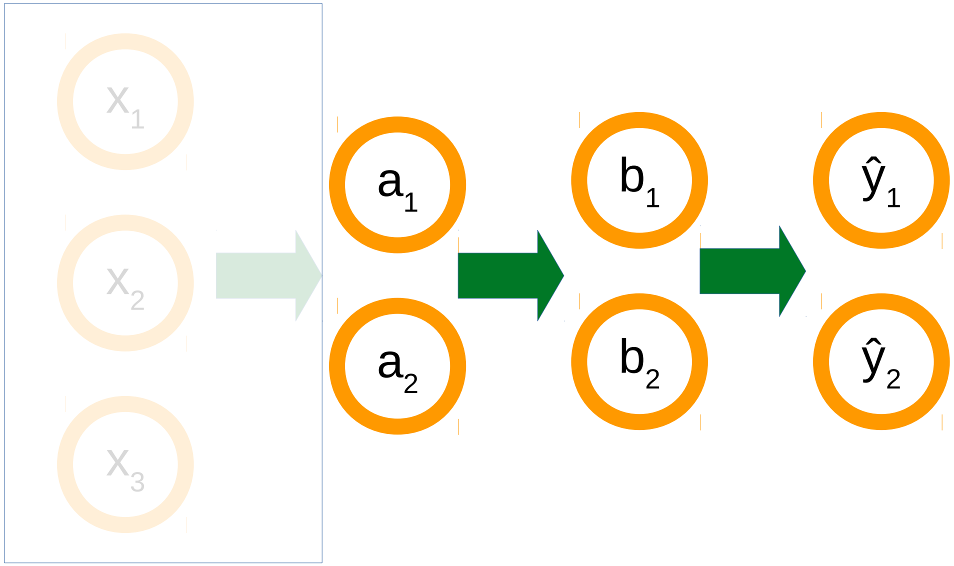
- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



- drop output layer
- consider hidden layer as new, dimension-reduced representation of input

Layer-wise training: AutoEncoder

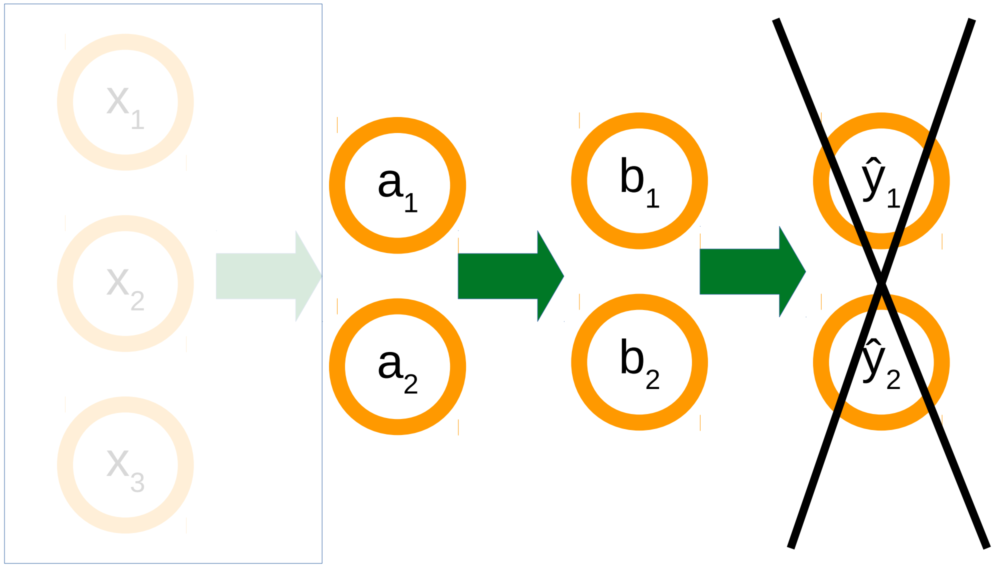
- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



- repeat procedure for next layer
- predict hidden layer activity
- $\hat{y}(x) \approx a$

Layer-wise training: AutoEncoder

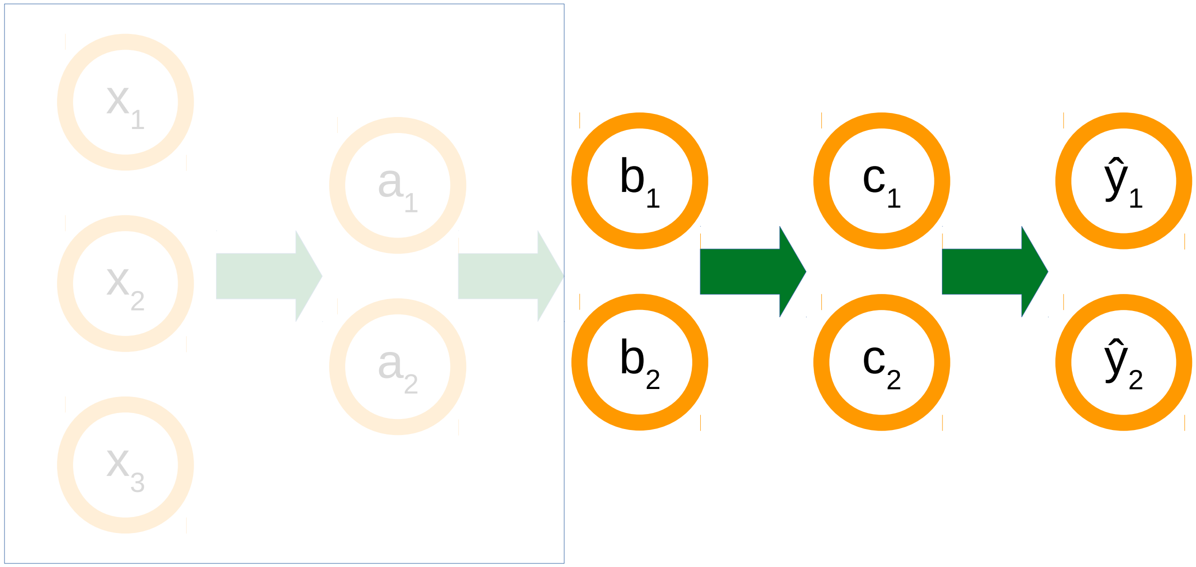
- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



- repeat procedure for next layer
- predict hidden layer activity
- $\hat{y}(x) \approx a$

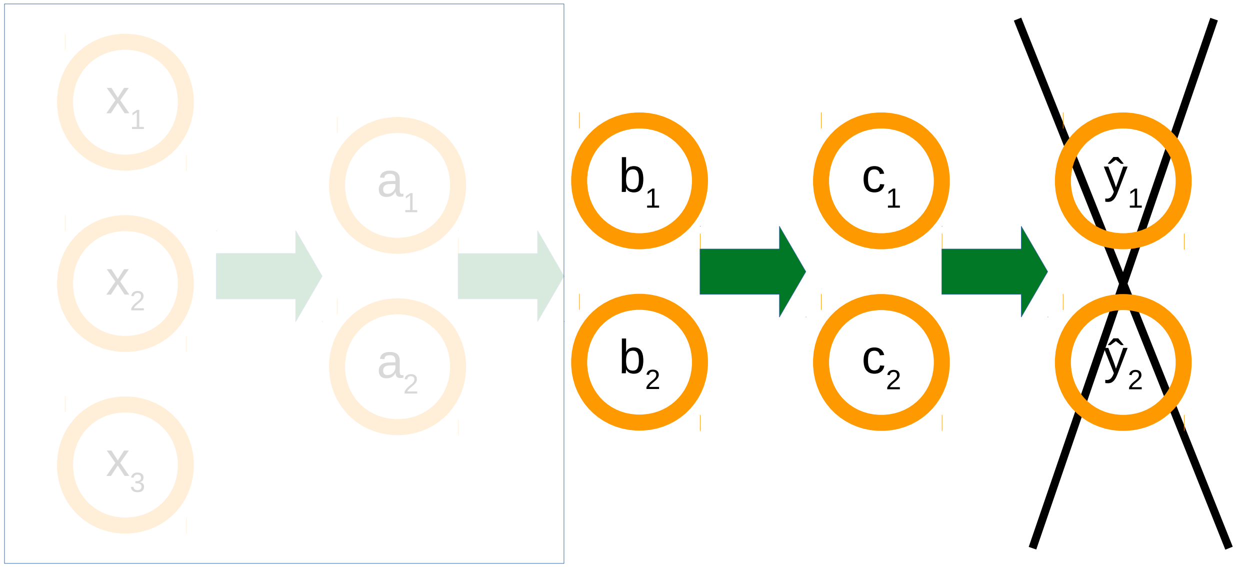
Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



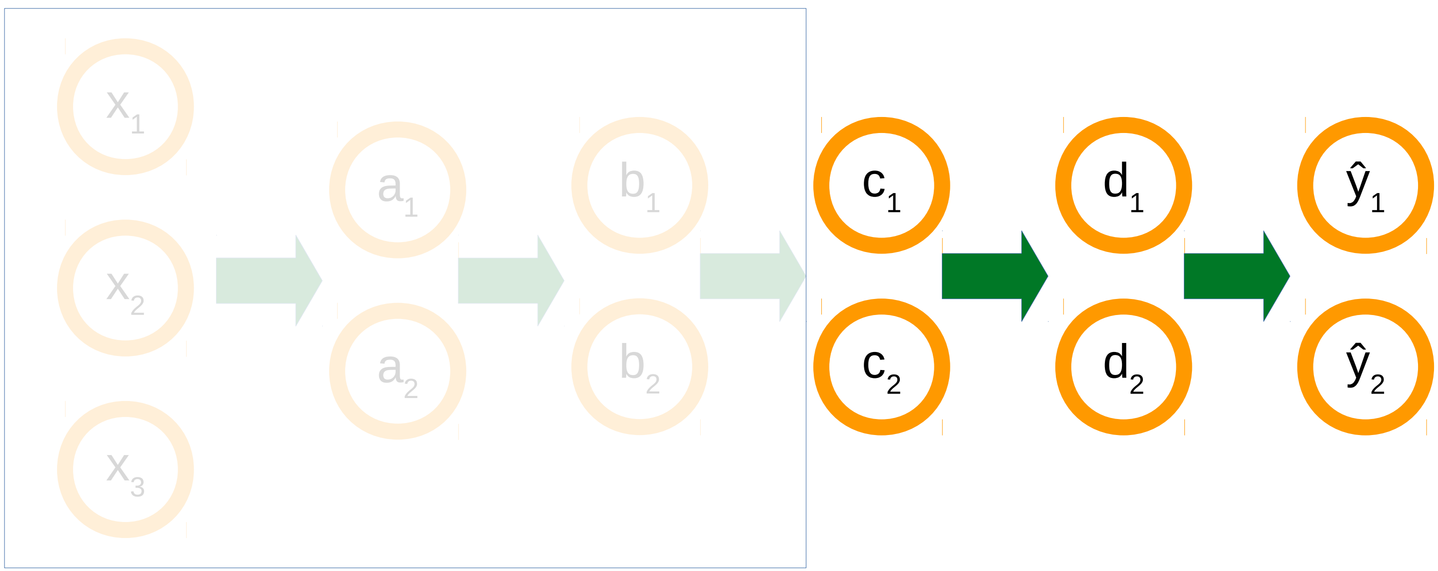
Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



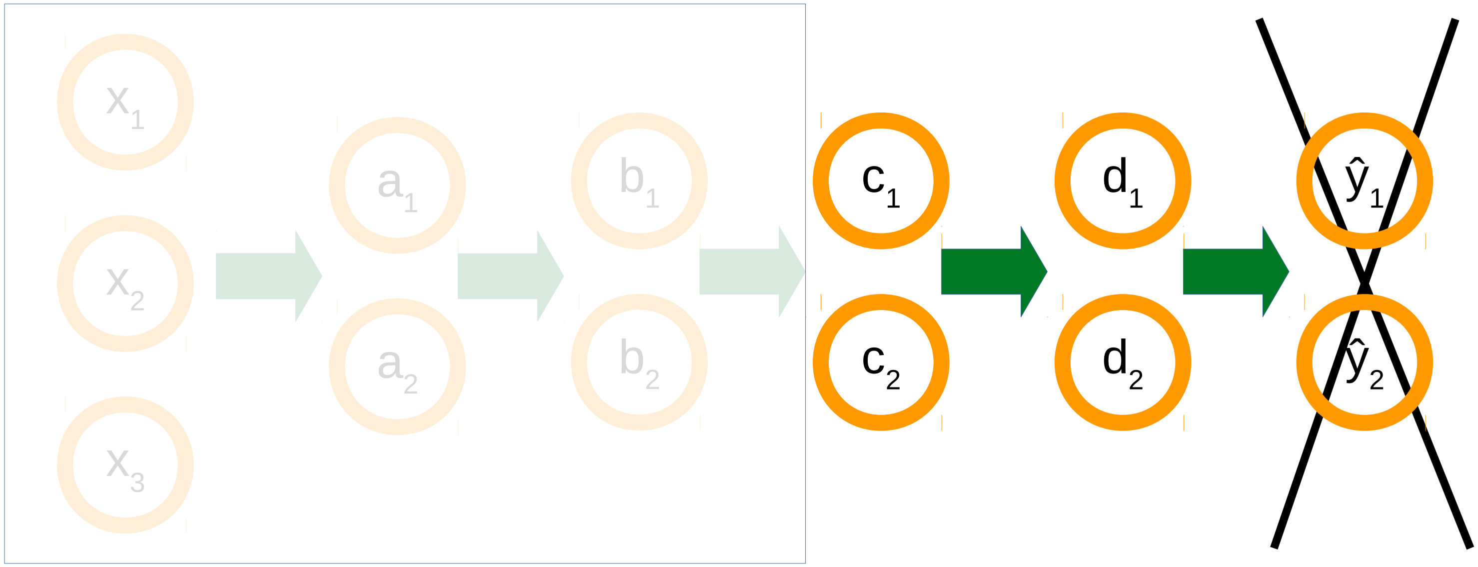
Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



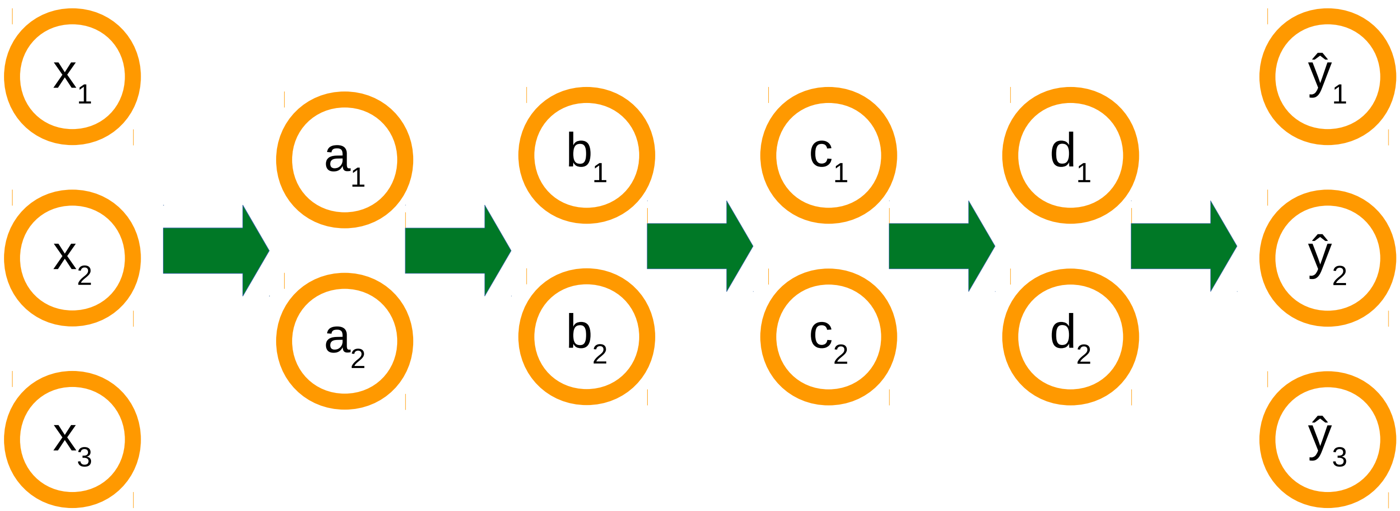
Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



Layer-wise training: AutoEncoder

- defeat vanishing gradient problem
- train network layer-wise using classical auto-encoder



- final supervised training to task

Denoising AutoEncoder

- stochastically corrupt input
- task: reconstruct original input
  - $E = \sum_{\alpha} \|\hat{y}(\tilde{x}^{\alpha}) - x^{\alpha}\|^2$

Denoising AutoEncoder

- stochastically corrupt input
- task: reconstruct original input
  - $E = \sum_{\alpha} \|\hat{y}(\tilde{x}^{\alpha}) - x^{\alpha}\|^2$
  - random dropout with probability p: $\tilde{x} = \begin{cases} 0 & \text{if } p < 0.5 \\ x & \text{else} \end{cases}$

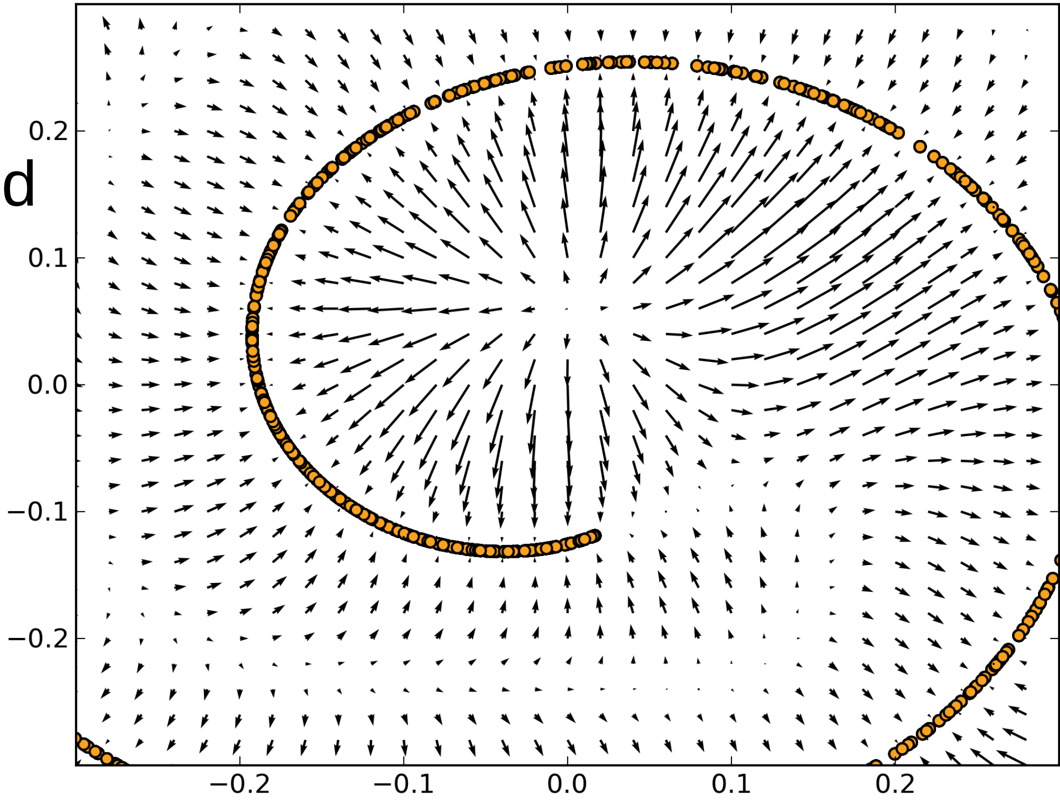
Denoising AutoEncoder

- stochastically corrupt input
- task: reconstruct original input
  - $E = \sum_{\alpha} \|\hat{y}(\tilde{x}^{\alpha}) - x^{\alpha}\|^2$
  - random dropout with probability p: $\tilde{x} = \begin{cases} 0 & \text{if } p < 0.5 \\ x & \text{else} \end{cases}$
  - Gaussian white noise: $\tilde{x} = x + \eta$

    $\eta \sim \mathcal{N}(0, \sigma)$

Denoising AutoEncoder

- stochastically corrupt input
- task: reconstruct original input
- learns vector field
  pointing towards
  data distribution manifold
- better generalization



Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- **boosting gradient descent**
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)
- Big Data

Boosting Gradient Descent

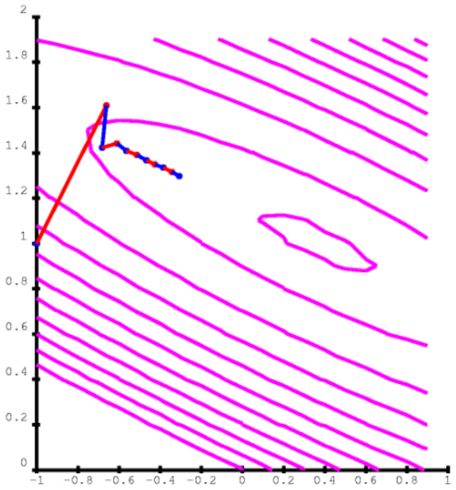
- Batching
- Momentum
- Learning Rate adaptation

Gradient Descent

- $E = \sum_{\alpha} E^{\alpha}$ $\qquad E^{\alpha} = (\hat{y}(x^{\alpha}) - y^{\alpha})^2$

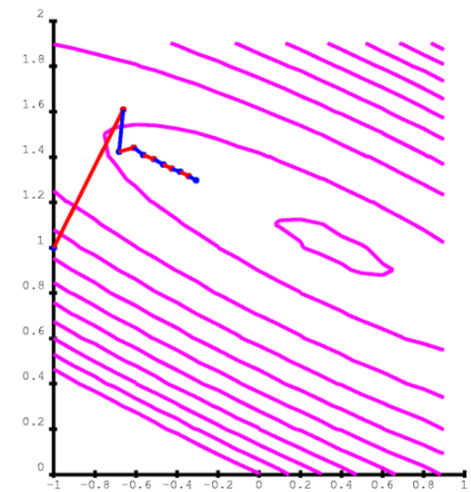
Gradient Descent

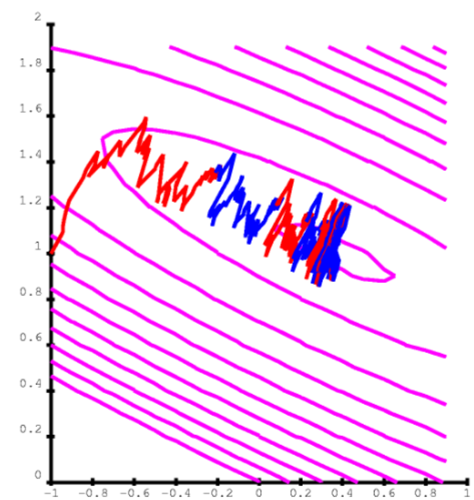
- $E = \sum_{\alpha} E^{\alpha} \qquad E^{\alpha} = (\hat{y}(x^{\alpha}) - y^{\alpha})^2$
- batch gradient: $\Delta w = -\eta \nabla_w E$
  - slow (full sweep over data required)
  - accurate



Gradient Descent

- $E = \sum_\alpha E^\alpha \qquad E^\alpha = (\hat{y}(x^\alpha) - y^\alpha)^2$
- batch gradient: $\Delta w = -\eta \nabla_w E$
  - slow (full sweep over data required)
  - accurate
- stochastic gradient: $\Delta w^\alpha = -\eta \nabla_w E^\alpha$
  - fast progress $\qquad \Delta w \propto \langle \Delta w^\alpha \rangle_\alpha$
  - fluctuates near minima / saddles
  - can escape from local minima



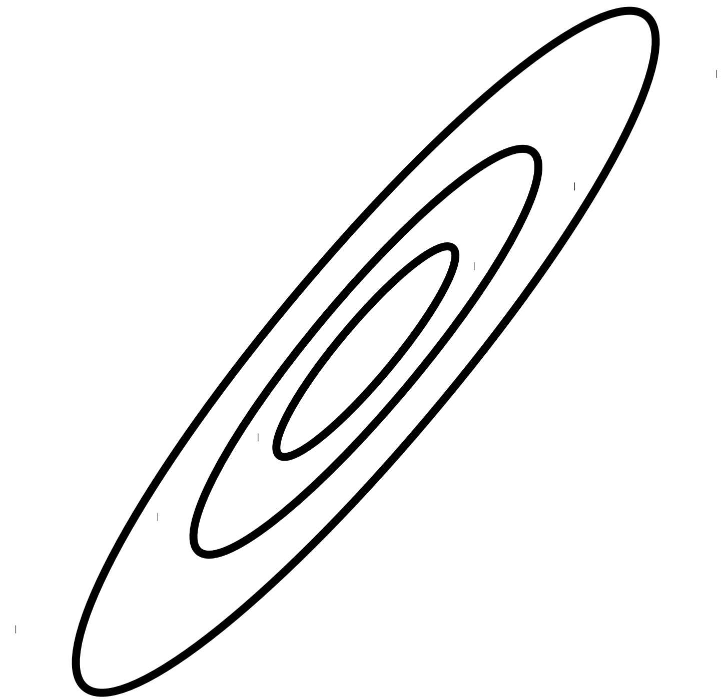


Mini-Batching

- combine the best of both worlds:
  average over small batch sizes
- fast convergence + reduced fluctuations
- assumes homogenous batches (e.g. randomly drawn)
- efficient on GPUs:
  parallel processing of several samples simultaneously
- reshuffle batches between epochs!

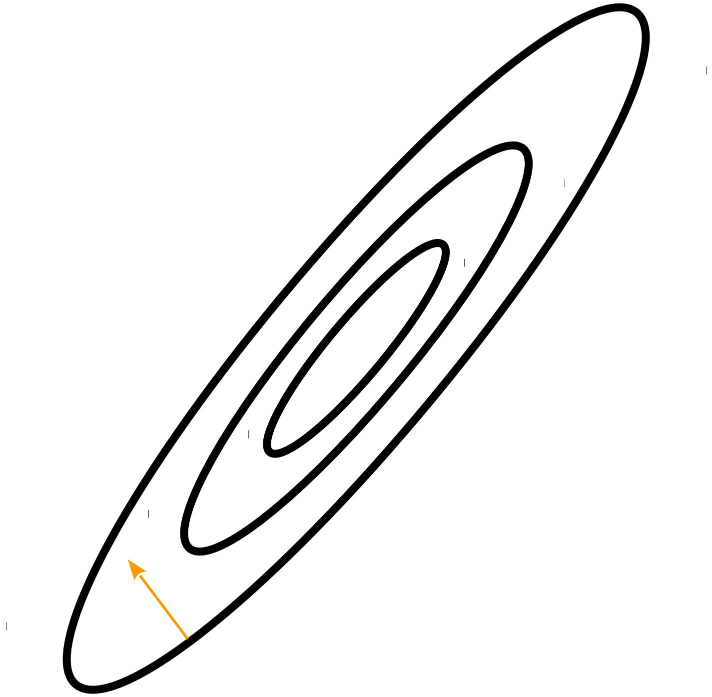
Classical Momentum

- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E$



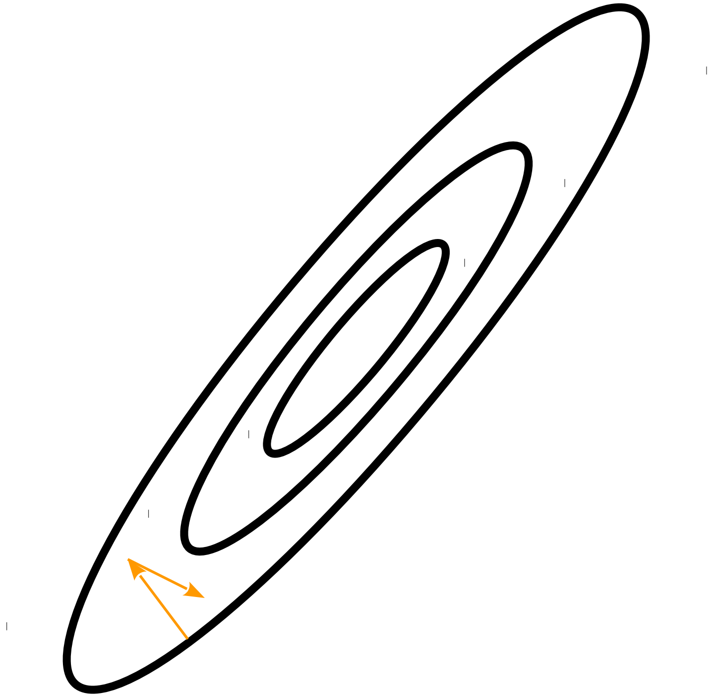
Classical Momentum

- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E$



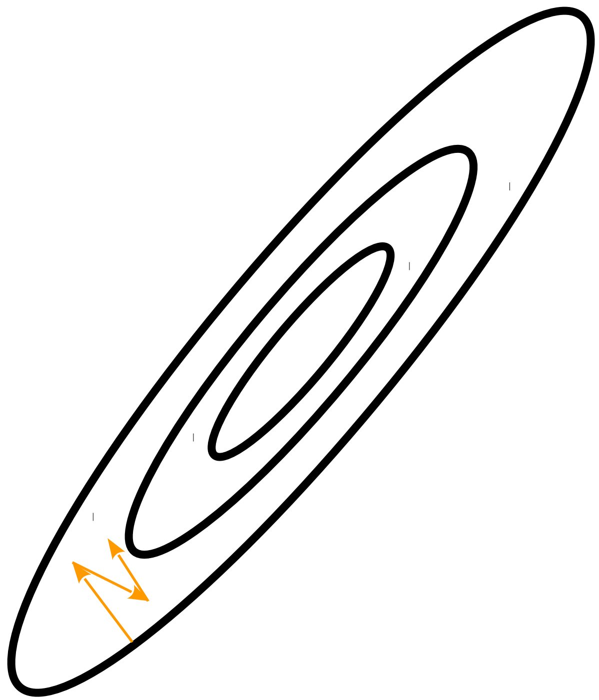
Classical Momentum

- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E$



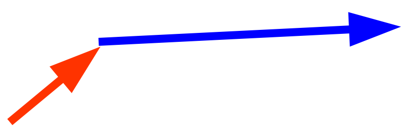
Classical Momentum

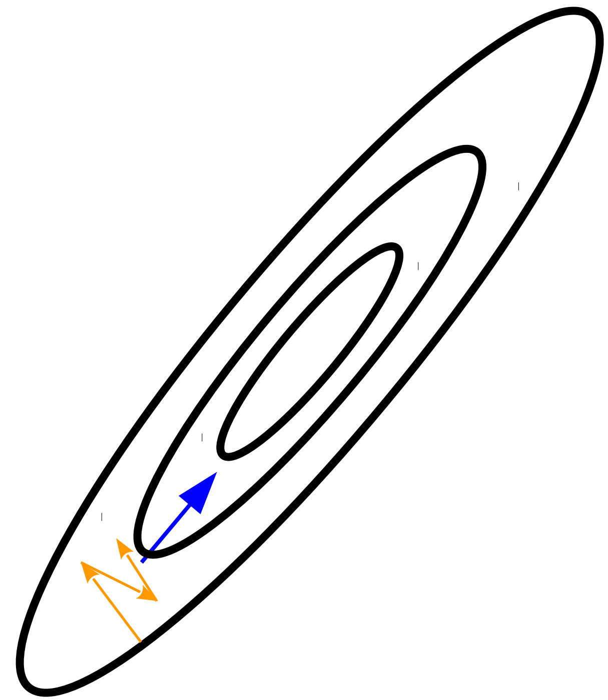
- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E$



Classical Momentum

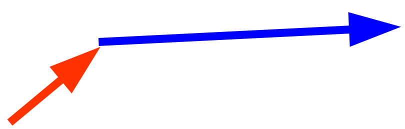
- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E + \alpha \Delta w_{t-1}$
- add discounted average gradient

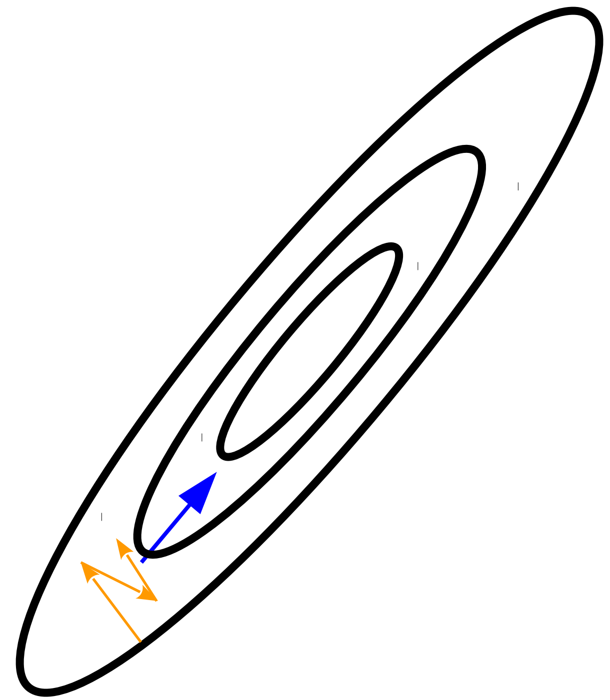




Classical Momentum

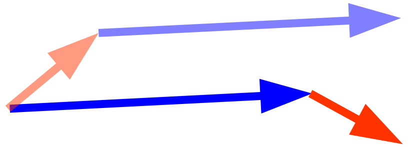
- gradient oscillates when navigating ravines
- $\Delta w_t = -\eta \nabla_w E + \alpha \Delta w_{t-1}$
- add discounted average gradient
- $\alpha \approx 0.9 < 1$
- speed-up by factor $\dfrac{1}{1-\alpha}$





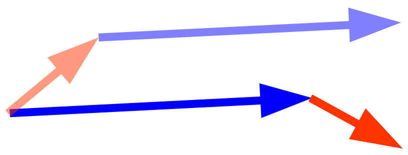
Nesterov-Momentum

- *invert order of momentum & gradient computation*
- **first** jump to new location (due to momentum)
- and **then** compute corrective gradient



Nesterov-Momentum

- *invert order of momentum & gradient computation*
- **first** jump to new location (due to momentum)
- and **then** compute corrective gradient
- It‘s better to correct a mistake after you have made it.

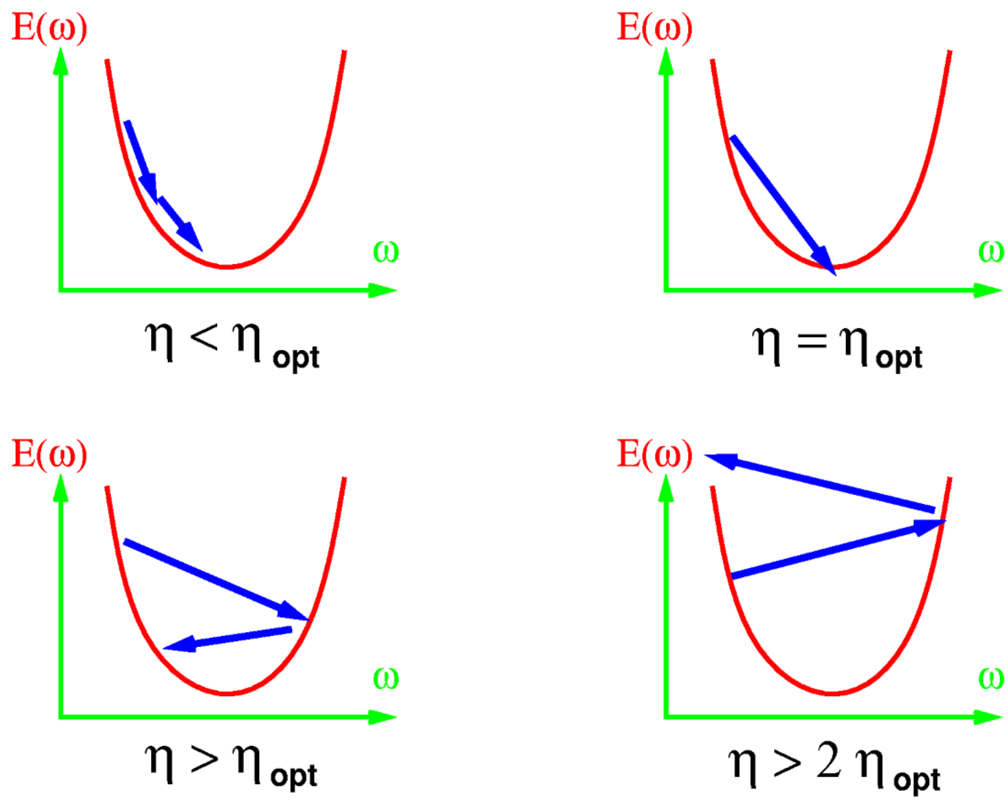


Learning Rate Adaptation

- gradient defines direction
- optimal step size depends on curvature

$$\eta_{\text{opt}} = \left(\frac{\partial^2 E}{\partial w^2} \right)^{-1}$$

- adapt η



Figs. from DL-tutorial @ NIPS2015

Resilient Backpropagation (RPROP)

- use individual learning rates η_i

η_i

$$\Delta w_i = -\eta_i \operatorname{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \operatorname{sgn}\left(g_i\right)$$

Riedmiller, Braun ICANN 1993

Resilient Backpropagation (RPROP)

- use individual learning rates η_i

η_i

$$\Delta w_i = -\eta_i \operatorname{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \operatorname{sgn}\left(g_i\right)$$

Riedmiller, Braun ICANN 1993

Resilient Backpropagation (RPROP)

- use individual learning rates η_i
- monitor direction (sign) of gradient g_i

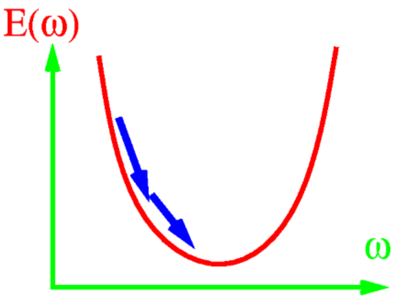
η_i

$$\Delta w_i = -\eta_i \operatorname{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \operatorname{sgn}\left(g_i\right)$$

Riedmiller, Braun ICANN 1993

Resilient Backpropagation (RPROP)

- use individual learning rates η_i
- monitor direction (sign) of gradient g_i
  - same sign: increase learning rate $(\cdot\eta^+)$



η_i

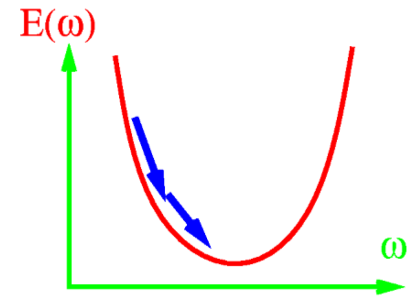
$$\Delta w_i = -\eta_i \operatorname{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \operatorname{sgn}(g_i)$$

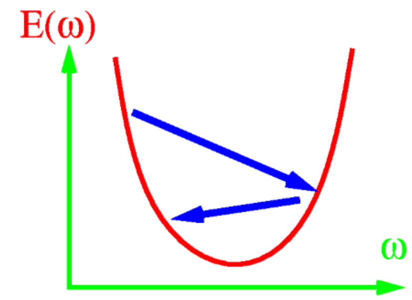
Riedmiller, Braun ICANN 1993

Resilient Backpropagation (RPROP)

- use individual learning rates η_i
- monitor direction (sign) of gradient g_i
  - same sign: increase learning rate $(\cdot\eta^+)$
  - sign change: decrease rate $(\cdot\eta^-)$
- use η_i directly as step size

$$\Delta w_i = -\eta_i \operatorname{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \operatorname{sgn}(g_i)$$



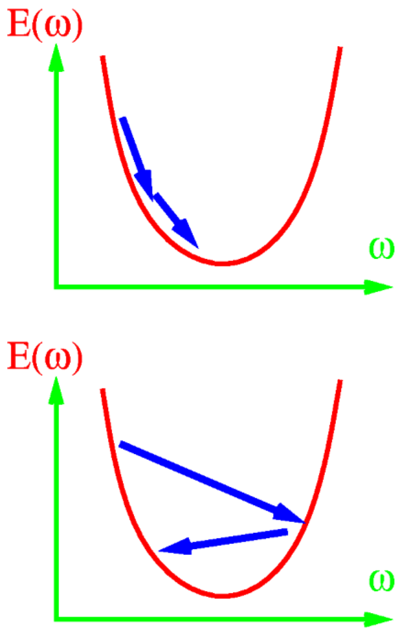


Riedmiller, Braun ICANN 1993

Resilient Backpropagation (RPROP)

- use individual learning rates η_i
- monitor direction (sign) of gradient g_i
  - same sign: increase learning rate $(\cdot\eta^+)$
  - sign change: decrease rate $(\cdot\eta^-)$
- use η_i directly as step size

$$\Delta w_i = -\eta_i \,\mathrm{sgn}\left(\frac{\partial E}{\partial w_i}\right) = -\eta_i \,\mathrm{sgn}\left(g_i\right)$$



- tends to overfitting

Riedmiller, Braun ICANN 1993

ADAGRAD

- automatically tune down learning rate based on learning history
- $$\Delta w_i(t) = -\frac{\eta}{\sqrt{\bar{g}_i^2(t) + \varepsilon}} \cdot g_i(t)$$

$$\bar{g}_i^2(t) = \sum_{\tau=0}^{t} g_i^2(\tau)$$

- denominator grows with past update steps
- effective learning rate tends to zero

➔ learning stagnates

ADADELTA

- average gradient updates across *finite* window using sliding average:

  $$\bar{g}_i^2(t) = \gamma \cdot \bar{g}_i^2(t-1) + (1-\gamma) \cdot g_i^2(t)$$

-

ADADELTA

- average gradient updates across *finite* window using sliding average:

$$\bar{g}_i^2(t) = \gamma \cdot \bar{g}_i^2(t-1) + (1-\gamma) \cdot g_i^2(t)$$

- $\Delta w_i(t) = -\dfrac{\eta}{\sqrt{\bar{g}_i^2(t) + \varepsilon}} \cdot g_i(t)$

ADADELTA

- average gradient updates across *finite* window using sliding average:

$$\bar{g}_i^2(t) = \gamma \cdot \bar{g}_i^2(t-1) + (1-\gamma) \cdot g_i^2(t)$$

- $$\Delta w_i(t) = -\frac{\eta}{\sqrt{\bar{g}_i^2(t) + \varepsilon}} \cdot g_i(t)$$

- correct units: nominator = average of weight updates

$$\Delta w_i(t) = -\frac{\sqrt{\overline{\Delta w}_i(t)^2 + \varepsilon}}{\sqrt{\bar{g}_i^2(t) + \varepsilon}} \cdot g_i(t)$$

$$\overline{\Delta w}_i^2(t) = \gamma \cdot \overline{\Delta w}_i^2(t-1) + (1-\gamma) \cdot \Delta w_i^2(t)$$

Adaptive Moment Estimation (ADAM)

- integrate momentum:
  sliding average of 1st and 2nd moments
- $m_i(t) = \gamma_m \cdot m_i(t-1) + (1-\gamma_m) \cdot g_i(t)$
- $v_i(t) = \gamma_v \cdot v_i(t-1) + (1-\gamma_v) \cdot g_i^2(t)$

Adaptive Moment Estimation (ADAM)

- integrate momentum:
  sliding average of 1st and 2nd moments
- $m_i(t) = \gamma_m \cdot m_i(t-1) + (1-\gamma_m) \cdot g_i(t)$
- $v_i(t) = \gamma_v \cdot v_i(t-1) + (1-\gamma_v) \cdot g_i^2(t)$
- biased towards zero (due to initialization)
  bias correction:

$$\hat{m}_i(t) = \frac{m_i(t)}{1-\beta_m^t} \qquad \hat{v}_i(t) = \frac{v_i(t)}{1-\beta_v^t}$$

-

Adaptive Moment Estimation (ADAM)

- integrate momentum:
  sliding average of 1st and 2nd moments
- $m_i(t) = \gamma_m \cdot m_i(t-1) + (1-\gamma_m) \cdot g_i(t)$
- $v_i(t) = \gamma_v \cdot v_i(t-1) + (1-\gamma_v) \cdot g_i^2(t)$
- biased towards zero (due to initialization)
  bias correction:

$$\hat{m}_i(t) = \frac{m_i(t)}{1-\beta_m^t} \qquad \hat{v}_i(t) = \frac{v_i(t)}{1-\beta_v^t}$$

- $\Delta w_i(t) = -\eta \cdot \frac{\hat{m}_i(t)}{\sqrt{\hat{v}_i(t)+\varepsilon}}$

Comparison of Optimizers

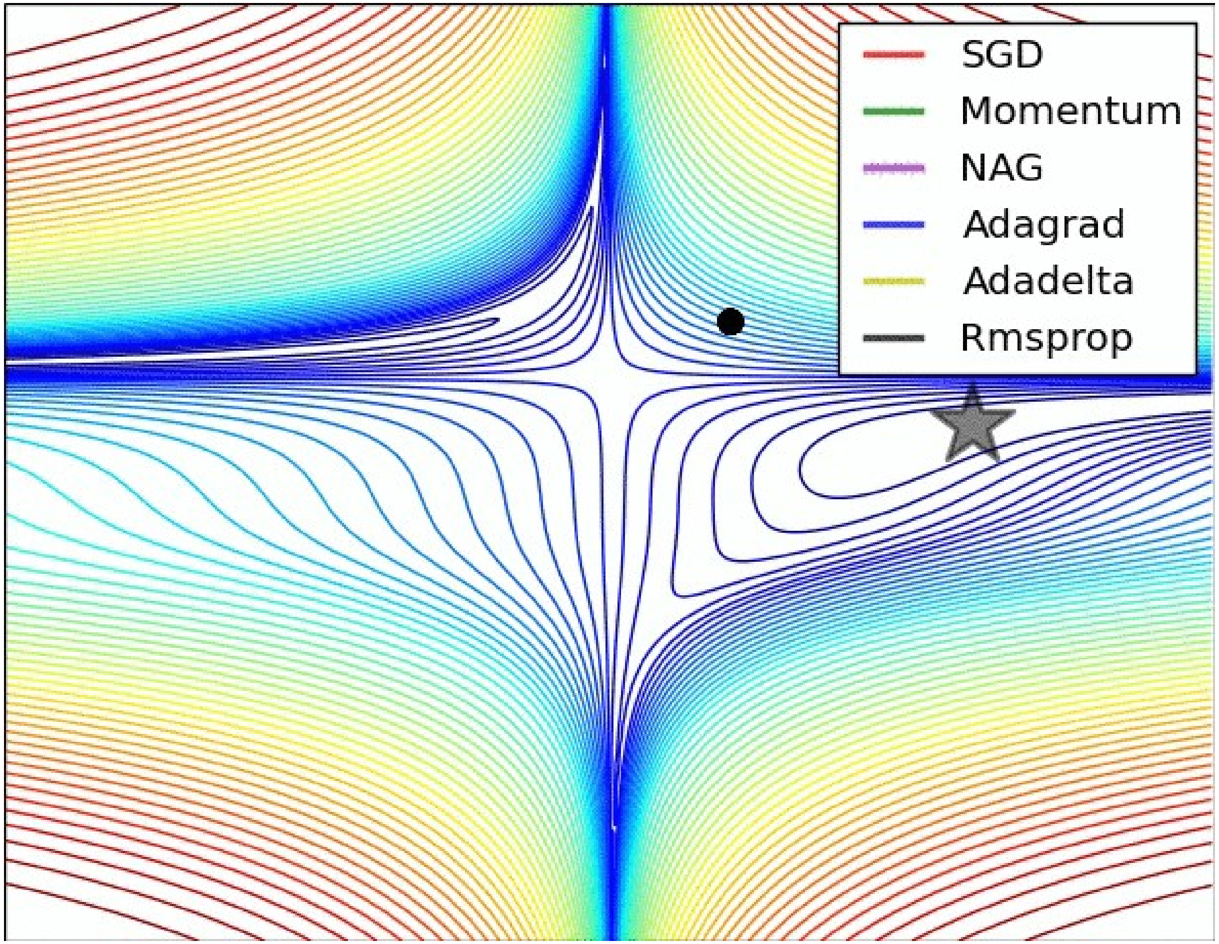


Fig. Sebastian Ruder

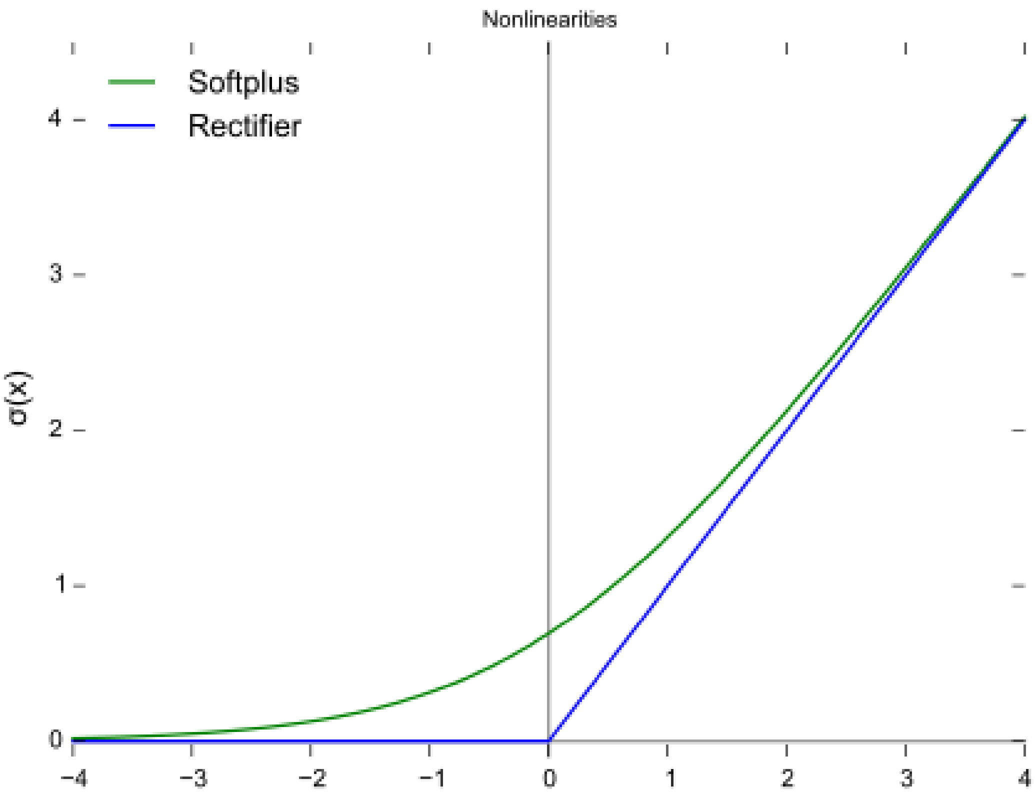
Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent
- **computing power**
  - simple non-linearity
  - highly-parallel processing (GPU)
- Big Data

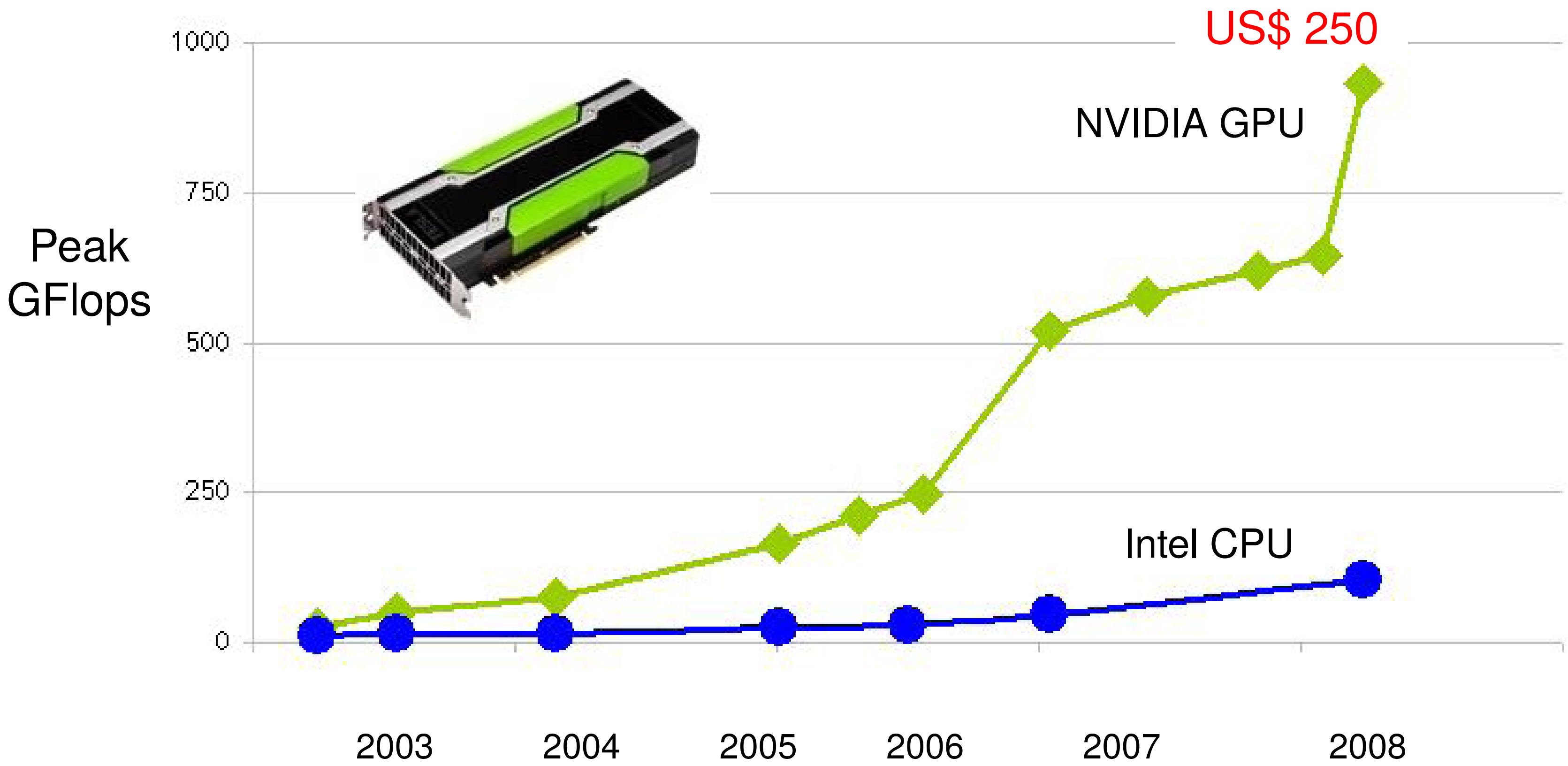
Rectified Linear Unit

- Softplus: $f(x) = \ln(1 + e^x)$
- ReLu: $f(x) = \max(0, x)$

- suitable to model real numbers
- max induces sparsity in hidden units
- no vanishing gradient



Highly Parallel Processing with GPUs



Ingredients for Successful Deep Learning

- powerful priors to reduce number of parameters
  - deep hierarchies
  - Convolutional Networks
- layer-wise training
- boosting gradient descent
- computing power
  - simple non-linearity
  - highly-parallel processing (GPU)
- **Big Data**

Big Data

- many model parameters (weights) require many training examples to avoid overfitting
- ImageNet: 1.3 million images
- unsupervised pre-training possible